Efficient Address Auto-Configuration in Ad hoc Networks - Protocol & Algorithms

Dissertation zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)

vorgelegt der Fakultät für Informatik und Automatisierung der Technischen Universität Ilmenau

von Dipl.-Ing. Ausama Yousef

1. Gutachter: Prof. Dr.-Ing. habil. A. Mitschele-Thiel
2. Gutachter: Prof. Dr. rar. nat. J. Seitz
3. Gutachter: Prof. Dr. rar. nat. O. Waldhorst

Tag der Einreichung: 14.10.2016
Tag der wissenschaftlichen Aussprache: 31.05.2017

urn:nbn:de:gbv:ilm1-2017000720

This page intentionally left blank

Acknowledgement

First, I would like to thank my advisor Prof. Mitschele-Thiel for his support and consistent academic guidance, which significantly helped me in accomplishing this thesis. I would like also to thank Prof. Seitz and Prof. Waldhorst for being co-supervisor of the thesis.

Second, I would like to thank all members of the Integrated Communication Systems group for their involvements in discussions and verifications of the thesis.

Finally, I would like to thank my family, especially my father and mother, for their encouragement and support.

Abstract

Mobile Ad hoc NETworks (MANETs) are an important part of mobile communications as they allow communications without the presence of an infrastructure. A MANET consists of an autonomous system of mobile devices. In contrast with infrastructure networks, MANET nodes act as hosts as well as routers. In the Internet, multi-hop communications are supported by the network layer, i.e. the Internet Protocol (IP). However, this requires the availability of a unique IP address. Due to the dynamic and decentralized nature of MANETs, and especially due to the mobility of nodes, providing and maintaining this unique IP address automatically in a decentralized way is a challenge addressed by auto-configuration protocols as part of the network layer. Several protocols to support this in fully decentralized environments as present in MANETs have been developed, e.g., the MANETConf, Buddy and Prophet protocols. However, they fail to solve the problem efficiently in scenarios where the nodes are highly mobile, e.g., as is the case with typical car-to-car applications.

In highly dynamic scenarios, for instance where two nodes may be connected just for a couple of seconds, the address allocation process has to be very fast. In addition, limited bandwidth and the high number of configuration actions in MANETs with high mobility require reducing protocol overhead to a minimum. Although there are several auto-configuration protocols developed for MANETs, they suffer from slow address assignment, high protocol overhead and address conflicts when multiple networks merge.

This thesis presents an address auto-configuration protocol that efficiently supports highly dynamic mobile ad hoc networks. This protocol, the Logical Hierarchical Addressing (LHA) protocol, focuses on the fast assigning of IP addresses to new nodes joining a MANET while minimizing the signaling overhead. Besides this, LHA introduces a solution for the merging problem ensuring, the uniqueness of IP addresses in the network when two previously independent MANETs merge. LHA is based on the idea that the address assignments can be achieved locally by the neigh-boring nodes of a requester, which in turn leads to a fast address assignment. Basically, in LHA, each configured node in a MANET is able to select, allocate and assign a unique address to a new node requesting an address that is free. By dividing the address space logically among configured nodes LHA is able to build a number of hierarchical structures of IP addresses. By this means, LHA solves efficiently the merging problem. Furthermore, the utilization of a certain assignment algorithm and specific address data structures is the key that LHA is able to solve the problem of missing IP addresses due to the departure of nodes. Because LHA is less dependent on unicast connections it reduces the signaling overhead and achieves fast address assignment. This in turn makes LHA highly suitable to the use in a wide range of scenarios, especially in those which are high mobility.

The work on which this thesis is based has shown that LHA is capable of fast address assignment with minimum signaling overhead even at high speeds of mobile devices. It outperforms MANETConf, Buddy and Prophet, which do not meet absolutely all of these requirements. Specific results of comparative studies for this thesis, based on ns2 simulations, show that MANETConf has a higher address configuration latency that do Buddy, LHA and Prophet. Comparing the assignment latency for LHA, Buddy and Prophet, results show that LHA is up to 86% and 57% faster than Buddy and Prophet respectively in high-density networks. In low-density networks LHA is up to 77% faster than Buddy but Prophet outperforms LHA, where it is 54% faster. However, this outperformance by Prophet in low-density networks has to be paid for high signaling overhead, wherein, LHA reduces the number of packets sent per time by up to 72% and 61% compared to the Buddy protocol and Prophet respectively in high-density networks and up to 58% and 44% in low-density networks respectively.

When the impact of the speed of mobile nodes is considered, it is found that LHA performs with signaling cost significantly better than Buddy and Prophet. The average number of messages transmitted by LHA is approximately 50% fewer than the number in Buddy and 30% fewer than in Prophet. With regard to average address assignment latency, LHA is up to 80% faster than Buddy in all mobile scenarios and it is up to 40 % faster than Prophet in high-speed scenarios. Although Prophet is faster than LHA in low-speed scenarios, the latency differences in those scenarios are small, with a satisfactory time of under 50 msec. With respect to the successful assignment process, LHA has a better performance in all mobile scenarios than that of Prophet or Buddy, wherein it shows a success rate of 100% in most attempts.

Contrary to other developed protocols, LHA presents an efficient and robust solution for merging networks because every node detecting a merger is able to solve the address conflicts resulting from the merger by sending only one broadcast message. Depending on the LHA concept, the change of all network addresses is achieved uniformly, which in turn provides routing protocols in MANETs a seamless way to update their routing table. Thus, the possible interruption of ongoing communications due to network address changes can be avoided. Moreover, the merger algorithm is able to handle simultaneous merging of more than two networks. The simulation results from all attempts show successful merging in adequate time.

Zusammenfassung

Mobile Ad-Hoc-Netzwerke (MANETs) sind ein bedeutender Teil der Mobilkommunikation, da sie Kommunikation ohne das Vorhandensein von Infrastruktur erlauben. Ein MANET besteht aus einem autonomen System von mobilen Geräten. Im Gegensatz zu Infrastruktur-Netzwerken agieren MANET-Knoten als Host, ebenso wie als Router. Im Internet werden Multi-Hop-Kommunikationen durch den Netzwerk-Layer unterstützt, z.B. das Internet Protokoll (IP). Dies verlangt jedoch die Verfügbarkeit einer eindeutigen IP-Adresse. Wegen der dynamischen und dezentralen Natur von MANETs und besonders wegen der Mobilität der Knoten, ist die automatische, dezentrale Bereitstellung und Verwaltung dieser eindeutigen IP-Adresse eine Herausforderung, die durch Autokonfigurationsprotokolle als Teil des Netzwerk-Layers gelöst werden soll. Zur Unterstützung dieser dezentralen Umgebung, die durch MANET repräsentiert werden, wurden verschiedene Protokolle entwickelt, wie MANETConf, Buddy und Prophet. Allerdings verfehlen sie eine effiziente Lösung des Problems in Szenarien mit hoch-mobilen Knoten, wie z.B. bei typischen „Auto-zu-Auto"-Anwendungen.

In stark dynamischen Szenarien, wo zwei Knoten zum Beispiel nur für ein paar Sekunden verbunden sind, muss der Prozess der Adresszuweisung sehr schnell erfolgen. Zusätzlich erfordern die limitierte Bandbreite und die hohe Anzahl an Konfigurationsaktionen in MANETs mit hoher Mobilität die Reduzierung des Protokoll-Overheads auf ein Minimum. Obwohl verschiedene Autokonfigurationsprotokolle entwickelt wurden, leiden sie unter langsamer Adresszuweisung, hohem Protokoll-Overhead und Adresskonflikten, wenn sich mehrere Netzwerke vereinigen.

Die vorgestellte Arbeit präsentiert ein Adressenautokonfigurationsprotokoll, das hoch dynamische mobile Ad-Hoc-Netzwerke unterstützt. Dieses Protokoll, genannt „Logical Hierarchical Addressing (LHA)", konzentriert sich auf die schnelle Zuweisung von IP-Adressen für neue Knoten, die einem MANET beitreten, und minimiert gleichzeitig den Signal-Overhead. Zusätzlich stellt LHA eine Lösung zum Vereinigungsproblem von Netzwerken vor und sichert die Eindeutigkeit von IP-Adressen, wenn sich 2 vorher unabhängige MANETs vereinigen. LHA basiert auf der Idee, dass die Adresszuweisung lokal durch jeden benachbarten Knoten eines anfragenden Knotens durchgeführt werden kann, was zusätzlich zu einer schnelleren Adresszuweisung führt. In LHA kann jeder konfigurierte Knoten in einem MANET für einen neuen Knoten eine eindeutige, freie Adresse auswählen und zuweisen. Durch die logische Aufteilung des Adressbereiches zwischen den konfigurierten Knoten kann LHA eine Anzahl hierarchischer Strukturen von IP-Adressen aufbauen, wodurch LHA das Vereinigungsproblem effektiv löst. Des Weiteren ist der Einsatz eines speziellen Zuweisungsalgorithmus und spezieller Adressdatenstrukturen der Schlüssel dafür, das LHA das Problem der durch das Verschwinden von Knoten fehlenden IP-Adressen lösen kann. Da LHA weniger abhängig von Unicast-Verbindungen ist, reduziert es den Signal-Overhead und erreicht eine schnelle Adresszuweisung. Dieser Effekt bewirkt die hohe Eignung von LHA für eine Vielzahl von Szenarien, insbesondere hoch mobile Umgebungen.

Die Untersuchungen, auf der diese Arbeit beruhen, haben gezeigt, dass LHA fähig zur schnellen Adressvergabe bei minimalem Signalisierungsaufwand ist, das gilt auch bei hohen Geschwindigkeiten von mobilen Geräten. Es stellt sich besser dar, als MANETConf, Buddy und Prophet, die alle diese Anforderungen absolut nicht erfüllen. Spezifische Ergebnisse von Vergleichsstudien für diese Arbeit (basierend auf ns2 Simulationen) zeigen, dass MANETConf eine höhere Adresskonfigurationslatenz als Buddy, LHA und Prophet hat. Der Vergleich von Ergebnissen der Zuweisungslatenz für LHA, Buddy und Prophet zeigt, dass LHA bis zu 86% schneller als Buddy und 57% schneller als Prophet in High-Density-Netzwerken ist. In Low-Density-Netzwerken verhält sich LHA bis zu 77% schneller als Buddy, aber Prophet übertrifft LHA, es ist in diesem Anwendungsfall 54% schneller. Diese Geschwindigkeit von Prophet in Low-Density-Netzwerken muss jedoch mit hohem Signalisierungsaufwand bezahlt werden, mit LHA verringert sich die Anzahl der Pakete bis zu jeweils 72% und 61% gegenüber dem Buddy-Protokoll und Prophet in High-Density-Netzwerken und um bis zu jeweils 58% und 44% in Low-Density-Netzwerken.

Wenn die Auswirkung der Geschwindigkeit der mobilen Knoten betrachtet wird, hat sich herausgestellt, dass der Signalisierungsaufwand von LHA deutlich geringer gegenüber Buddy und Prophet ist. Die durchschnittliche Anzahl der gesendeten Nachrichten bei LHA ist circa 50% geringer gegenüber Buddy und 30% geringer gegenüber Prophet. Im Hinblick auf die durchschnittliche Adresszuweisungslatenz ist LHA bis zu 80% schneller als Buddy in allen mobilen Szenarien und es ist bis zu 40% schneller als Prophet in High-Speed-Szenarien. Obwohl Prophet schneller als LHA in Low-Speed-Szenarien ist, sind die Latenzunterschiede in diesen Szenarien klein und liegen bei einem guten Wert von unter 50 msec. Mit dem Nachweis des erfolgreich stattgefundenen Zuweisungsprozesses hat LHA eine bessere Leistung in allen mobilen Szenarien gegenüber Prophet oder Buddy, wobei es eine Erfolgsrate von 100% in den meisten Versuchen zeigt.

Im Gegensatz zu anderen entwickelten Protokollen präsentiert LHA eine effiziente und robuste Lösung für Netzwerk-Vereinigungen, da jeder Knoten, der eine Zusammenführung von Netzwerken entdeckt, nur eine Broadcast-Nachricht schicken muss, um alle resultierenden Adresskonflikte im Netzwerk zu lösen. Gemäß dem LHA-Konzept findet eine einheitliche Änderung aller Netzwerkadressen statt, was wiederum zu einem nahtlosen Weg zur Aktualisierung der Routing-Tabelle für Routing-Protokolle in MANETs führt. Damit kann eine mögliche Unterbrechung der laufenden Kommunikation aufgrund von Netzwerkadressänderungen vermieden werden. Darüber hinaus ist der Vereinigungsalgorithmus in LHA auch in der Lage, die gleichzeitige Zusammenführung von mehr als zwei Netzwerken zu realisieren. Die Simulationsergebnisse aus allen Versuchen zeigen einen erfolgreichen Vereinigungsprozess in einer adäquaten Zeit.

Table of Contents

Abbreviations

4G	*fourth Generation network*

A

AA	*Address Agent*
ABA	*Agent Based Addressing*
ACK	*Acknowledgment*
AIPAC	*Automatic IP Address Configuration*
AODV	*Ad Hoc on Demand Distance Vector*

B

BER	*Bit Error Rate*
BOOTP	*Bootstrap protocol*
BRAN	*Broadband radio access networks*

C

CBA	*Cluster-Based Address auto-configuration*
CIDR	*Classless Inter-Domain Routing*
CoReS	*Configuration and Registration Scheme*

D

DACP	*Dynamic Address Configuration Protocol*
DAD	*Duplicate Address Detection*

DAPRPA	*Defense Advanced Research Project Agency*
DHAPM	*Dynamic Host Auto-configuration Protocol for MANETs*
DHCP	*Dynamic Host Configuration Protocol*
DSR	*Dynamic Source Routing*

E

ESTI	*European Telecommunications Standards Institute*

F

FAACP	*Filter-based Address Autoconfiguration Protocol*
FAP	*Filter-based Addressing Protocol*
FCC	*Federal Communications Commission*
FSR	*Fisheye State Routing*

G

GPS	*Global Positioning System*
GSM	*Global Standard for Mobile communication*

I

IETF	*Internet Engineering Task Force*
IoT	*Internet of things*
IP	*Internet Protocol*
IPv4	*Internet Protocol version 4*
IPv6	*Internet Protocol version 6*
IANA	*Internet Assigned Numbers Authority*

L

LHA	*Logical Hierarchical Addressing*
Layer 3	*network Layer of TCP/IP model*
LTE	*Long Term Evolution*
LOS	*Line of Sight*
LHA	*Logical Hierarchical Addressing protocol*
MANET	*Mobile Ad hoc NETwork*

M

MAC	*Media Access Control*
MANET WG	*MANET Working Group*
MN	*Merger Networks*
MTU	*Maximum Transmission Unit*

N

nMNet	*number of Merging Networks*
ns2	*network simulator 2*

O

OLSR	*Optimized Link State Routing*

P

PAA	*Primary Address Authority*
PACMAN	*Passive Auto-configuration for Mobile Ad-hoc Networks*
PDA	*Personal Digital Assistant*
PDAD	*Passive DAD*
PRNet	*Packet Radio Networks*

Q

QDAD	*Query-based DAD*

R

RTC	*Real-Time Communications*

U

UAVs	*Unique IP Address Verification agents*
UML	*Unified Modeling Language*
UMTS	*Universal Mobile Telecommunications Systems*
U-NII	*Unlicensed-National Information Infrastructure*

W

WDAD	*Weak DAD*
WG	*Working Group*
WiMAX	*Worldwide Interoperability for Microwave Access*
WLAN	*Wireless Local Area Networks*

Z

ZEROCONF	*Zero Configuration Networking*

Chapter 1 Introduction

Mobile Ad hoc NETworks (MANETs) are infrastructure-less, distributed and self-organized networks used to cover geographical parts where communication infrastructure does not exist. Such areas might be mountainous or hilly or dotted with isolated, small villages. For that purpose, the MANET features networks have certain advantages over the traditional wireless communication networks, for instance, LTE[1], UMTS[2], GSM[3], WiMAX[4] or WLAN[5]. Some of the main advantages as presented in [1] can be summarized as follows:

- **Flexibility:** Because MANETs can be constructed in a short time (built on-the-fly), they have higher mobility and flexibility than conventional wireless networks.
- **Scalability:** While the communications in traditional wireless communication networks are conditional on the existence of the Line of Sight (LOS), the multi-hop feature in MANETs makes communication beyond the LOS possible at high frequencies.
- **Efficiency:** The use of short multi-hop communications in MANETs instead of long-distance node to central base station communication has another advantage over traditional wireless networks in that it reduces interference levels and increases spectrum reuse efficiency. This in turn makes it possible to use unlicensed unregulated frequency bands.
- **Economy:** MANETs can in some cases be more economical, as they eliminate fixed infrastructure costs and reduce power consumption at mobile nodes.

The advantages of MANETs bring with them a variety of applications and opportunities [2]. Ad hoc networks were initially a military application and emerged from the Defense Advanced Research Project Agency (DAPRPA) [3] [4]. Today, however, they are used to make users' lives easier or safer, as in the case of meeting rooms (interactive lectures) or automotive networks (e.g. the BMW talking cars project for local hazard warning [5]). The major recommendation for MANETs now is that they are the only possible solution to enable communications in disaster areas, where communication infrastructure has been destroyed. Another field where the MANETs are essential is the future Internet, it has developed into what is today known as "the Internet of things (IoT)" [6] . This future vision of the Internet assumes that each object (thing) is embedded with sensors and is able to communicate. Of course, the communication is not planned to

[1] LTE: Long Term Evolution (LTE). It is the standard of fourth generation (4G) mobile communication networks.
[2] UMTS: the Universal Mobile Telecommunication System (UMTS). It is the standard of third generation (3G) mobile communication networks.
[3] GSM: the Global Standard for Mobile communication (GSM). It is the European standard of second generation (2G) mobile communication networks.
[4] WiMAX: the Worldwide Interoperability for Microwave Access (WiMAX).
[5] WLAN: Wireless Local Area Network.

be infrastructure-based. Although ad hoc wireless communication networks represent a promising solution for the various shortcomings of infrastructure networks, there are many constraints which make it difficult to design and develop protocols for MANETs. Some of those constraints as presented in [7] can be summarized as follows:

- **Power supply:** In MANETs the power is a critical term because batteries carried by each mobile node cannot live for long. This in turn limits the processing power, services and applications that can be supported by each node.
- **Node capabilities:** Due to the differences found in the wireless cards, drivers, and antennae characteristics with which the ad hoc node may be equipped there will be great heterogeneity in node radio capabilities in MANETs. This leads to varying transmission/receiving capabilities. In addition, processing capabilities differ from one device to another because mobile devices might have different software/hardware configurations.
- **Resource limitations:** As defined by the Federal Communications Commission (FCC) and European Telecommunications Standards Institute (ESTI), ad hoc networks and many other devices, such as wireless phones and microwaves, generally operate in ISM (Industrial, Scientific, and Medical) and U-NII (Unlicensed-National Information Infrastructure) bands. Because ISM and U-NII bands are a scarce medium and because there are many devices sharing the medium, ad hoc networks have fewer available frequencies. This in turn leads to high interference and medium contention.
- **Asymmetric links:** Radio irregularity, a common phenomenon of the wireless medium, is due to multiple factors, such as variance in RF sending power and different path losses depending on the direction of propagation. Such radio irregularity, along with interference, obstacles and noise level difference, contributes to the existence of asymmetric links [8] in MANETs. These link quality differences are a serious problem for communication protocols.
- **Limited bandwidth:** Ad hoc wireless networks inherit the traditional problems of wireless communications which operate with limited bandwidth. This means that only a limited amount of information can be transmitted over a certain period of time. Due to the high rate of collisions and *Bit Error Rate* (BER) in MANETs, the use of maximum packet size is inefficient in achieving high throughput in data transmissions. The simulation in [9] and empirical study in [7] show that a proper packet size, i.e. *Maximum Transmission Unit* (MTU), of ad hoc applications should be set to approximately 512 byte.
- **Dynamic topology:** Because ad hoc nodes are free to move randomly and organize themselves arbitrarily, the wireless topology of a network may change rapidly and unpredictably. Such dynamic changing of network topologies results in route changes, frequent network partitions and mergers, which turn network configuration and management into a very hard task.

1.1 Problem Statements

Basically, MANETs are autonomous systems of mobile devices (nodes) which are capable of working as both hosts and routers. This enables mobile nodes to communicate with each other through multiple hops without any need for a predefined communication infrastructure, see Figure 1-1. In the figure there are three ad hoc nodes (A, B and C) and each of them has a limited transmission range which enables the node to communicate with its neighbour directly (direct connection). In this way, A can communicate only with B because they are the only nodes within each other's transmission range. On other hand, if A needs to communicate with C it has to use a multi-hop communication (indirect connection). This means the intermediate node B should work as router between A and C. A has to send its packets firstly to B and then B will deliver (route) them to C. The routing protocols in MANETs [10] have been a main focus of the MANET Working Group (MANET WG) [11] which works to standardize the ad hoc protocols. Routing from the source to the destination nodes is, however, impossible as long as the node is not assigned a certain identifier [12], i.e. it is necessary to have a unique addresses assigned to every node. Because MANETs are IP-based and must adhere to the main features of the IP world, each mobile node must be configured with a unique IP address before it can join a network [13].

The dynamic and decentralized nature of MANETs means that a manual configuration of mobile nodes with unique IP addresses is impracticable. So, the issue of an automatic addressing process is managed by specific address auto-configuration protocols.

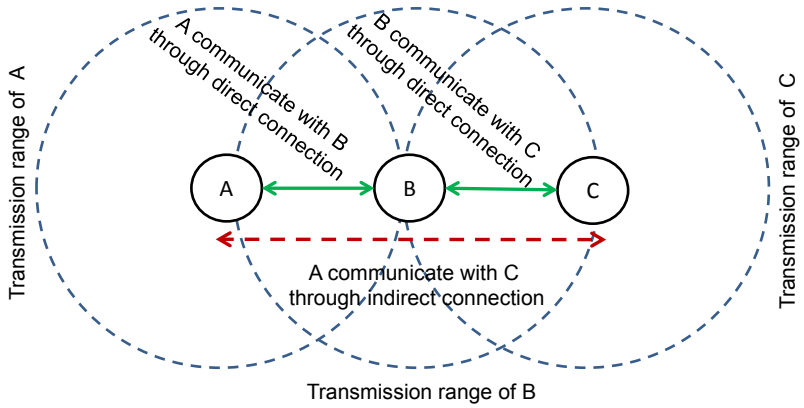


Figure 1-1 : Multi-hop connections in MANETs

In such an auto-configuration process, the first task a new joining node has to accomplish is the acquisition of a unique IP address. Thus, the address assignment process is considered a main issue of every auto-configuration protocol. In this context the following main challenges have to be faced:

- **Address assignment latency:** this latency is defined as the latency a joining node requires to obtain a unique IP address. It is crucial and it must be as short as possible, despite the fact that the acquisition of a unique IP address from a high dynamic network with no central management is not an easy job and will in normal situations take a considerable amount of time, even some seconds. This delay deteriorates, for sure, ongoing applications and may force established sessions to be reset or closed. It must be remembered that the address assignment process is not needed only when the node switches on and joins the network for the first time. The task may, for instance, be carried out in response to movements of the node from one ad hoc network to another one during an active session. The problem gets even more complicated when the network grows in size.
- **Address assignment overhead:** the IP address assignment process demands recent knowledge of IP addresses currently utilized by the nodes existing in the network. Hence, the collection of such knowledge in MANETs is usually achieved by broadcasting[1] control messages inside the network, with the consequence of a considerable overhead in the address assignment. This overhead is, of course, not desired and must be minimized, since it wastes radio resources. Note that the larger the network, the more the overhead. The broadcast itself implies other problems, also, that cannot be neglected, those of collisions and contentions as described in [14].

There is yet another issue in automatic address configuration that must be considered, namely the management of IP addresses within the MANET. The well-known problem in this context is the possible existence of duplicated addresses, resulting when two different MANETs merge or a new joining node is assigned an IP address which is already in use. This latter case is a possibility if the available IP addresses are not properly managed. It must be remembered that the available space of IP addresses in MANETs is limited and nodes are free to join and leave. So, IP addresses of disjoining (departing) nodes can be reused. Otherwise, an inadequate handling of departing nodes may lead to a depletion of the available address pool.

The dynamic nature of MANETs gives rise to yet more potential problems, such as limited bandwidth, symmetric/asymmetric links, network scalability, an ever-changing topology due to node mobility. All in all, the handling of IP addressing problems is very complicated. Therefore, the traditional addressing protocols known from infrastructure-based networks (e.g. the Dynamic Host Configuration Protocol (DHCP) [15], ZeroConf [16], IPv6 stateless auto-configuration [17], etc.), are inadequate to the needs of MANETs because they support neither decentralized management nor multi-hop communication. This has led to the development of many address auto-configuration protocols to cope with the dynamic nature of MANETs. However, all the solutions so far suffer from a range of problems which include slow address assignment process, large protocol overhead, potential address conflicts during merging operations and others. This thesis is intended as a contribution to the further development of address auto-configuration protocols so that they better meet the challenges here described.

[1] Broadcasting is one of the main operations in the MANETs, such as detecting the neighbors and paging a particular host, sending an alarm signal, route discovery in source initiated and on-demand routing, announcement, …etc. [7]

1.2 Auto-configuration Requirements

An adequate address auto-configuration mechanism must, to be adequate to the main challenges already mentioned, include functions which will properly handle two main issues. These are address assignment and address maintenance. Address assignment is concerned with the selection, allocation and assignment of IP addresses to newly joining nodes, while address maintenance manages the available IP address pool, ensures that IP addresses in use are unique and releases any unused addresses. The following provides more insight into what is required in both address assignment and maintenance solutions.

- **Address assignment**: any solution for the address assignment problem has to fulfill the following requirements:
    - *Self-assignment*: any new node should be able to construct a new Ad hoc network in case it does not find an existing MANET. A self-assignment of IP addresses must be supported.
    - *Duplicate-free address assignment*: this means the assignment of a definitely not in-use (i.e. unique) IP address to a new joining node.
    - *Multi-hop assignment support*: because MANETs are multi-hop networks, any address assignment solution should be capable of communicating over multiple hops when necessary.
    - *Efficient assignment process:*
        - Minimized address assignment latency: this implies that the time required to allocate and assign a new IP address should be minimized to avoid communication disruption.
        - Minimized signaling overhead: the number of protocol messages exchanged to complete the assignment process should be reduced to a minimum.
- **Address maintenance**: any solution aiming at a proper and efficient maintenance of IP addresses of MANETs must fulfill the following requirements:
    - *Proper handling of departing nodes*: to properly handle the IP address pool, departing nodes must be detected as quickly as possible, so that their addresses can be reused. Moreover, the responsibility of managing the released addresses should be given to a predefined node. In this way, it should be possible for address conflicts arising from simultaneous assigning of a released address to different requesters to be avoided.
    - *Proper detection of missing nodes*: because the detection of abruptly departing nodes needs adequate cooperation among nodes, efficient methods of keeping the degree of negotiation among network nodes low are required.

- *Proper handling of network partitions*: because the nature of MANETs is dynamic, an ad hoc network may split into two or more partitions. After the network partitioning, new nodes may join the partitions, thus, acquiring IP addresses. The assignment process in each partition does not take account of the addresses in other partitions. The problem arises when these partitions merge again later to form a single ad hoc network, since duplicated addresses may arise.
- *Proper handling of network merge*: usually a merger occurs when two neighboring networks move into the other's transmission range. As a result, duplicate addresses may exist and these may cause erroneous routing when conflicts are detected and an attempt is made to solve them. The merger function should, therefore, deal with such cases as fast as possible with low control overhead. Moreover, special cases such as a simultaneous multiple network merge, in which two networks merge at the same time, should be considered in any solution.

1.3 Dissertation Objectives

Consistently with the statements in the previous sections, the main objective of this dissertation is the development of an address auto-configuration protocol for MANETs. The intention is that the protocol fulfils the requirements mentioned above and advances the state of art by considering the following:

- **Protocol Efficiency** – achievement of fast operation with low cost.
- **High reliability** despite the operation failures like address duplication for unpredicted reasons. So, the protocol developed must be able to recover from operation failures.
- **Scalability and capability** of coping with the dynamic nature of MANETs.
- **Robustness** against the dropping of control messages since message dropping might otherwise produce address conflicts.
- **Suitability** for a wide range of scenarios which will include simultaneous merging of more than two networks or the standing alone of a partitioned node.

1.4 Contribution

The following contributions have been accomplished in the work here described:

- **An in-depth analysis of the address auto-configuration issue** in MANETs.

- **A comprehensive review previously available address auto-configuration protocols** along with a discussion of the pros and cons of each protocol. In this context, the dissertation classifies existing protocols and provides a qualitative comparison between those described.
- **Development of an address auto-configuration protocol** named Logical Hierarchical Addressing (LHA), see Chapter 3. The proposed protocol guarantees the uniqueness of assigned IP addresses during the assigning phase (i.e. a stateful protocol, see Section 2.2.2 for more details). It is designed to achieve the assignment process faster than other stateful protocols with minimal signalling overhead. Moreover, in this thesis it is shown that LHA is robust against control message dropping and also reliable, since merging and partitioning of MANETs have been considered during the designing of the LHA protocol.

1.5 Thesis Structure

The rest of the document is organised as follows. Chapter 2 gives an overview of the existing literature of address auto-configuration protocols for mobile ad hoc networks. Chapter 3 is a description of the basic idea and the main algorithms of the LHA protocol. An evaluation of the performance of LHA compared to other stateful protocols is provided in Chapter 4. ly, Chapter 5 summarizes the main points covered by the thesis and suggests future work in this field of research.

Chapter 2 Address Auto-Configuration

This chapter deals mainly with the state of the art in the auto-configuration process studies. The chapter structure is in six sections of which Section 2.1 gives an overview of IP configuration in conventional infrastructure networks. Then, in Section 2.2, a classification of the addressing protocols used so far in MANETs is presented. To assist fine understanding the contribution made by this thesis, a new classification applicable to stateful addressing protocols is defined in Section 2.3. Section 2.4 presents new categories of possible mechanisms for resolving address conflicts in cases of merging networks, while in Section 2.5 new categories of possible network partitioning detection mechanisms used in MANETs are described. Then in Section 2.6, a comparative analysis of representative auto-configuration protocols for each of the newly classified stateful classes is introduced.

2.1 Early IP Address Auto-Configuration Efforts

In the 1980s, the main method of assigning IP addresses to hosts was manual, by administrators. As the number of communicating devices increased, allocating and managing IP addresses required a great deal of administrative work. To underpin growth, mainly in the Internet, the automatic administering of IP addresses was suggested. To serve network users in any infrastructure networking scenario different automatic address schemes were developed for infrastructure networks. As classified in [18] these address allocation schemes are divided into stateful and stateless schemes. When a network is using a stateful scheme the allocation responsibility for addresses is given to a certain entity which keeps the state information of all addresses in the network in a database. In contrast, the stateless schemes let each entity allocate its address by itself. To ensure the uniqueness of its address, the entity performs a procedure called Duplicate Address Detection (DAD).

2.1.1 Infrastructure Stateful IP Addressing

The main idea of protocols following the stateful method is that new nodes have to get their IP addresses from a certain node (central node) which has knowledge about every address state (allocated or free). Therefore, this method can by in very concept ensure the uniqueness of IP addresses in the network because the assigning node has a global overview of all allocated addresses. To neglect the type of environment (a network connection may be permanent or temporary), different protocols have been designed to allocate static or dynamic addresses. If IP ad-

dresses are dynamic, it means that an address may be changed in each connection. An address is called a static IP address if there is no need for the change in each connection. In the following, two protocols are presented as examples of the two.

2.1.1.1 BootP protocol

The Bootstrap protocol (BOOTP), presented in [19], is intended for a comparatively static environment in which every network connection is permanent. A node can apply this protocol to find out its own IP address, a gateway address and the address of a server host from BOOTP server. The basic allocation operation of BOOTP is that the host broadcasts a 'bootrequest' packet on the local network by using the limited broadcast IP address 255.255.255.255 or the server's IP address (if known). The BOOTP server then answers with a 'bootreply' packet, which contains the requester's IP address, the router's IP address, etc. The main disadvantages of BOOTP:

- No local configuration for each host is provided, but only central configuration.
- The host's configuration parameters are stable for long time, despite the fact that the host is only active for a certain time. Therefore, the protocol is not appropriate for a dynamic address assignment.

2.1.1.2 DHCP protocol

In contrast to the BOOTP protocol, the *Dynamic Host Configuration Protocol* (DHCP) is designed to work in dynamic enviroments where network connections change frequently. As a successor of BOOTP, the DHCP protocol, of which there are two versions [15] and [20] for IPv4[1] and IPv6[2] respectively, is widely used in Ethernets and Wireless LANs to configure the host's IP address. Mainlay, DHCP distinguishes between three types of IP address allocation, as follows:

- Automatic allocation where a permanent IP address is assigned to a host.
- Dynamic allocation where a temporary IP address is assigned to a host.
- Manual allocation where a network administrator assigns an IP address to a host.

As with BOOTP, the basic allocation operation in DHCP is built on a client-server model, where the server refers to the assigning node (DHCP server) and the client refers to a node (DHCP client) requesting an IP address. In this model, four messages are needed to allocate an address to every client. Initially, the client broadcasts[3] a DHCPDISCOVER message on its local physical subnet. Upon receipt of this message, each DHCP server responds with a DHCPOFFER message that includes an available network address. If two or more of these messages are received by the client, it has to choose a certain server and it broadcasts a DHCPREQUEST message including the 'server identifier' of the selected server. After that, the server selected in the DHCPREQUEST message responds with a DHCPACK message containing the configuration

[1] Internet Protocol version 4 (IPv4) [90], which provides a number of IP addresses with a 32-bit number
[2] Internet Protocol version 6 (IPv6) [92], which provides a number of IP addresses with a 128-bit number
[3] For IP broadcast address, it is defined in [96] to set all bits of its local host part to one.

parameters for the requesting client. The configuration process is complete when the client receives this message including the information sent before in the DHCPOFFER message. To achieve dynamic allocation in DHCP, the address is assigned for a certain time called "lease" to the client. In order to use the address for a longer time the client has to extend its lease with subsequent requests by sending of DHCPREQUEST messages. This means that a client has to send this message periodically to be able to continue using a particular network address. The main disadvantages of DHCP:

- Like BOOTP, DHCP offers only a central management system.
- DHCP is not intended for use in configuring routers.
- DHCP has high protocol overhead due to the lease algorithm.

2.1.2 Infrastructure Stateless IP Addressing

The stateless method enables each new node joining the network to select its IP address by itself. There is thus a high likelihood of address conflicts due to the absence of information about the used addresses in the network. Therefore, a negotiation mechanism between the new host and the other hosts in the network is needed. Basically, protocols following this mechanism are called Zero Configuration Protocols. The task of developing Zero configuration protocols was given to IETF Zeroconf WG's whose main goal is to enable direct communication between two or more computing devices via IP [21]. A host in zero configuration protocols is able to configure itself with a local IP address which can be used only for communication with other devices connected to the same physical (or logical) link. The local address can be selected randomly or by using an identifier of the interface, as the following examples show.

2.1.2.1 IPv4 Link-Local addresses

In the protocol presented in [16], the dynamic configuration of a host is done by using IPv4 link-local address[1], wherein, the host selects randomly its address in the 169.254/16 prefix. Notice the address of this prefix is not routable. Therefore, to configure a host with unique addresses, the host needs to be able to detect and solve any collision among its neighbouring nodes. Mainly, such zero configuration protocols are designed to work satisfactorily over one hop connection networks such as home networks, automobile networks, airplane networks, or ad hoc networks at conferences, emergency relief stations, and many others [22]. Hence, such protocols are disadvantageous for normal ad hoc networks which work and support multi-hop connections.

2.1.2.2 IPv6 Link-Local addresses

It is possible for a host to configure itself with IPv6 local addresses as in [17] . This is done by appending the ID number of its interface (in most cases its MAC[2] address) to the well-known

[1] is proposed in [16] to perform only communications in the same physical (or logical) link when stable, routable addresses are not available (such as on ad hoc or isolated networks)
[2] A media access control address: "is a unique identifier assigned to network interfaces for communications at the data link layer"

link-local prefix [23] . In this way and in contrast to DHCP, the allocation of an address is achieved without asking contact with any kind of server. Because MAC addresses may not be unique [24], the host, as with IPv4 Link-local, has to verify that its selected address is unique among all neighbors on the host link by broadcasting NDP messages [25] to the link. If there is a conflict with any neighbor the host has to abort the process and manual configuration (by an administrator) of the interface is required. In principle, the administrator should be able to supply an alternate interface identifier that may override the shortcut. IPv6 link local still inherits the problem of IPv4 above wherein the address is available only over single hop.

2.1.3 Why do infrastructure solutions not suit ad hoc networks?

It is obvious from the above that the Internet protocols are not suitable for MANETs because they neither support decentralized management nor possess any mechanism to ensure uniqueness beyond the local (one-hop) connection of a node. It is this that is inspiring effort to modify the protocols and, indeed, to develop new protocols for MANETs which can barely exist without multi-hop connections. There follow the auto-configuration protocols developed for MANETs.

2.2 Classification of Auto-configuration Protocols

Various auto-configuration protocols have been developed to meet the requirements of ad hoc networks (doing without infrastructure-less and employing multi-hop connections). The authors in [26] present possible techniques in different auto-configuration steps, such as initial address, initial method, choosing of new address or allocation method. Because the address assignment task is the first step in all protocols and it also essential to solve the other address problems (conflicts in case of merger networks or the reuse of the addresses of departure nodes) the most of the classification effort so far has been focused on the methods of achieving the address allocation task.

A key point of the classification of addressing protocols, highlighted in [27], is the maintenance of address allocation tables in nodes because a network's overall knowledge of free and allocated IP addresses in a network should ensure the unique assignment of IP addresses. Thus, the author in [27] divides the protocols into three categories; stateful, stateless and hybrid classes. In the stateful class the assignment process in a network depends on two concepts; firstly, at least one configured node in the network must own the allocation table with up-to-date information or actual status of all allocated addresses in the network; secondly, a new node must get its unique address by requesting a free address from the configured node which owns the allocation table. In principle, the information on the allocation tables in the stateful class helps the configured nodes to ensure unique assignment of IP addresses to the requesting nodes. In contrast, the protocols of the stateless class are necessarily unable to guarantee the allocation of unique addresses. The new nodes, usually, allocate themselves new addresses without any knowledge of available free addresses in the network. The stateless method has the advantage of being much

faster than stateful one, which depends on other nodes in the network to achieve the allocation process. However, a successful assignment of unique addresses in stateful methods needs usually a synchronization mechanism to ensure that up-to-date information is maintained in the network. As a solution between the two methods, the hybrid class has been defined. Basically, a protocol of hybrid category combines both methods utilized in other categories, wherein a new node allocates itself with an address by using a table from neighboring nodes which does not include up-to-date information. In other words, the new node requests IP information from one of its neighbors which usually keeps past knowledge about the allocated addresses in the network. Depending on such information the new node selects its new address. In this case the likelihood of allocating unique addresses will be higher than in the stateless category but lower than in the stateful one.

In Kyriakos *et al.* [28] another method of classifying the auto-configuration protocols is introduced. This depends on the possible detection of conflicting addresses within the allocation process. In this classification three categories are named: conflict detection allocation, best effort allocation and conflict free allocation. Because there is no mention of the similarity between this classification and that in [27] the researchers may have been confused by the terminology. Therefore, the point is made here by way of disambiguation. Comparing the method of both classifications reveals that there are no differences between them because the main key in each is whether network information is used or not. For example, a protocol of conflict free allocation category enables the assignment of addresses without causing address conflicts in the network. Basically, this can be done only when this protocol possess information about the addresses of the whole network. Vice versa, if a protocol possesses information about all addresses used in the network (i.e. it belongs to the stateful category) it can ensure that the joining nodes can be assigned with unique addresses. Thus, the stateful, stateless and hybrid categories are similar to the conflict free allocation, conflict detection allocation and best effort allocation ones respectively.

The classification in [26] leaves some uncertainty in certain steps, e.g. in *initial method* any protocol of both classes (*directly* and *DAD*) should send a packet to verify the uniqueness of any selected address. On the other hand, due to the attributes counterpart of other classifications and, also, wide range of the concept of stateful and stateless protocols in most RFCs and works, such as [17], [20], [18], [29], [30], [31] and [32], the categories followed in this thesis are those used in [27]. In the following sections each class will be presented with examples.

2.2.1 Stateless protocols

In protocols following the stateless approach, the new node constructs its addresses by itself. These addresses are typically based on a hardware ID or on a random number given by a random generator. Although this mechanism, of course, is a fast assignment process, there is no guarantee that this address is unique in the network. So, if the number of nodes in a network is large and the address space is small, the probability of getting of address conflicts can be assumed to be very high in the network. To deal with the existence of conflicted addresses these protocols,

therefore, use a Duplicate Address Detection (DAD) [17] mechanism which is responsible for detecting the conflicts and, then, informing the responsible nodes so that they change their IP addresses. If DAD is utilized after a node has used its selected address in a running real-time application, this will violate the requirements of such applications and increase the control messages to solve possible address conflicts. However, in the case when DAD is used before the address configuration is finished it is subject to other constraints related to issues such as delay and reliability in MANETs.

Perkins *et al.* [33] and Fazio *et al.* [34] provide two examples of stateless protocols which employ DAD before a node is able to use its selected address in established communications. In the first protocol, devised by the MANET group of Internet Engineering Task Force (IETF), a new node chooses a new address by itself. Before using this address to communicate with other nodes in the network, it has to rule out an address collision by broadcasting a request to its own address. If the selected address is not in use by another node in the network the new node is able to use it in any communication with other nodes in the network, otherwise it has to select another address and initiate a new query process by itself. In contrast, the query process in the second protocol, Automatic IP Address Configuration (AIPAC), will not be done by the new node itself. Here, the new node (requester) selects from its neighboring nodes an initiator node which is already a configured node. The initiator node negotiates an address within the network on behalf of the requester. For the query process, the initiator node randomly selects an address from a predefined space and tests it with the network with the DAD procedure. The approach in both protocols is what is basically known as a Query-based DAD (QDAD). In this approach the address configuration takes longer than desired. Additionally, unreliably exchanged messages because of lost and dropped messages between the nodes may lead to incorrect decisions.

Even when the DAD mechanism is applied after the address has been assigned, as is the case with Weak DAD (WDAD) [35] and Passive DAD (PDAD) [36], the requirements of real time applications fail to be met, because the duplicated addresses lead to a change in the IP address of some nodes which may have ongoing Real-Time Communications (RTC) [37] . This exchange interrupts the transport and network layer connection, and the reconnection of them probably involves a long time and many control message exchanges. In WDAD the new node has to choose an initial IP address by itself. Then it should pick a random key by means of alternative methods. The final IP address can be composed out of the key and the initial IP address. This will reduce the probability of two nodes choosing the same IP address. If a node receives a packet containing an IP address that is stored in its routing table, but with a different key, an address conflict is detected. The key is only generated once by each node. There is a possibility that any two nodes with the same address choose the same key and the conflict will not be discovered in this case. In order to decrease the probability of such a situation, the key length has to be increased. This again results in routing protocol overhead. Afterwards, the DAD is used to detect if the same address already exists in the network. This will lead to the case which was explained above and violate the requirements of real time applications. In PDAD the new node chooses its address randomly. Then it can communicate with other ones in the network. Unlike WDAD, the PDAD mechanisms do not need to add additional information to the IP address, like keys. Therefore no protocol overhead is generated. Instead, every node in the network analyzes incoming routing protocol packets to discover the conflicted addresses. However, the analyzing process is

based mainly on the state information used by routing protocols. The optimal choice of PDAD mechanism is thus to use proactive routing protocols. Because the probability of conflicted addresses in this protocol is high, the control messages to solve all conflicts lead to signaling overhead; especially when more than one node discover a conflict. PDAD has the same problem as WDAD with real time applications.

2.2.2 Stateful protocols

The configuration process of stateful protocols depends mainly on the concept of the maintenance of a kind of allocation table which includes information concerning all allocated addresses in the network. In contrast with the stateless approach, by using a stateful approach the uniqueness of the address can be ensured by concept. However, there is a cost. The assignment process is slow, which does not suit the fast handoff needed for the interworking aspects in hybrid networks. There follow examples of different stateful protocols which use different kinds of allocation tables.

ABA (Agent Based Addressing) protocol in [38] uses the same principle as the DHCP protocol, utilizing a centralized allocation table. In this protocol only one node, the Address Agent (AA), is allowed to assign addresses to requesting nodes. The AA maintains the allocation table containing already assigned IP addresses, corresponding MAC addresses, and lifetimes. A synchronization process is needed to refresh the status of IP addresses in the allocation table. Therefore the AA periodically floods a “verify” message to all configured nodes in the network, asking them about their current status. Those receiving it must respond with reply messages - a process which adds protocol overhead. Additionally, due to unreliable message exchanges the allocation table status may be incorrect, causing duplicated addresses. Moreover, the main challenge is handling the crashed AA node because many nodes may try to be address agent simultaneously. This brings with it additional signaling cost, especially in high mobility scenarios.

As presented in [39] the MANETconf protocol distributes to all nodes in a network a common allocation table which contains the available address space in the network. It prevents the assignment of the same address to more than one node by maintaining an additional allocation table called pending allocation table. These allow every configured node in the network to assign a unique IP address to any requester. When a new node enters the network, it should select an initiator node from its neighbors. The initiator will be responsible for the configuring process. Therefore, the initiator selects an address that is neither in its allocation table nor in the pending allocation one. Then it floods a requesting message to all other configured nodes in the network. This message will inform all nodes that the selected address will be used by a new node. If there is a conflict with any node, the initiator receives a reply messages from these nodes. Then it selects another address and initiates another attempt to assign the new one. Until this process succeeds, much undesirable latency is added. This approach also needs a synchronization procedure to ensure that the allocation tables of the nodes are always up to date. Hence additional and reliable message exchange is needed among all nodes. These messages increase the overhead in the network.

In [40] the authors propose a Buddy protocol utilizing multiple disjoint allocation tables. This means that the global allocation table is split among all nodes. The splitting of the allocation table is based on the binary buddy algorithm known from memory management. A new node has to select one of its neighbors, which will help in the configuration process. This neighbor is called the initiator. The function of an initiator is to assign half of its allocation table, if available, to the requester without asking other nodes for permission. If the node does not have an allocation table, it asks the other nodes in the network to do the task. Upon receiving the table, the new node directly chooses its unique address from this table. Now the new node can also act as initiator for nodes joining in the future. This mechanism can assign a unique address to a new node faster than the other ones above. However, the mechanism suffers from some problems. One problem can arise if any of the nodes crashes. Then a part of the address space may be destroyed. As a solution, a synchronization procedure is needed to detect the holes in the address space. This again increases the protocol overhead. Another problem relates to merging and partitioning. When two networks with identical allocation tables merge, many conflicted addresses may be found in the network. In this case there is no efficient method which can be used to solve the problem.

2.2.3 Hybrid protocols

The hybrid approach uses elements from both stateless and stateful approaches. The protocols maintain allocation tables and use DAD mechanisms, such as the Passive Auto-configuration for Mobile Ad-hoc Networks (PACMAN) protocol [41]. A node which runs this protocol assigns an address to itself depending on an allocation table obtained from a neighboring node and a random algorithm. This mechanism reduces the duplicated address probability. But in the case where many nodes join the network simultaneously this probability increases. Conflicts arise because the allocation table is not up-to-date. In the approach a Passive DAD (PDAD) mechanism is employed to solve these conflicts. The node using PDAD detects the duplication by checking passively the ongoing communication in the network. This means that the duplicated addresses may not be discovered when a node is assigned by a new address. Moreover, this mechanism depends on ongoing routing protocols.

In Syed *et al.* [42], which is presented as the SAAMAN protocol in [43]), the assignment process needs two steps to ensure the allocation of a unique IP address to every node in the MANET. In the first step, the new node has to get a temporary IP which should be selected randomly from a node depending on Global Positioning System (GPS) information and has to use a DAD mechanism to avoid temporary address duplication. Basically, the authors expect each node in the network to be able to identify its position using GPS. The goal of using GPS information is to minimize the duplicate selection of temporary addresses. In this way, the network should be divided into a predefined number of squares and to each square is allocated a disjoint set of the whole temporary IP set. When the node selects its temporary IP, it has to go to the second step called real IP assignment. The node, here, has to select randomly an address from another set (real set) and it has to check the uniqueness of this address in the network by asking some nodes which are predefined as server nodes in the network. A similar principle is utilized

in Filter-based Address Auto-configuration Protocol (FAACP) [44] of which the server nodes are called Unique IP Address Verification agents (UAVs) and hold information of all allocated address in the network. As with PACMAN, this protocol is impacted by simultaneous joining of many nodes located on a single square. Moreover, it suffers from inaccurate allocation sets when the GPS information is missing due to device defect or environmental conditions, such as bad weather.

2.2.4 Why Stateful Protocols?

The protocols following the categories used in [27] are next presented, with a summary in Table 2-1 of main features of each class. In the table it is clear that stateless protocols possess the fastest mechanism of address assignment with lowest signaling cost. This is because, in stateless protocols, every new node can select and assign a new address by itself. However, there is a cost in the high probability of address conflicts in the network. The detecting and changing of duplicated addresses then presents other problem wherein the requirements of real-time applications are not met.

Table 2-1: Features of stateful, stateless and hybrid classes

Features	***Stateless***	***Stateful***	***Hybrid***
Assignment process	*Fast*	*Slow*	*Medium*
Signaling cost	*Low*	*High*	*Medium*
Address uniqueness	*Not guaranteed*	*Guaranteed*	*Best effort*
Main focus	*DAD mechanism*	*Reduce latency & signaling cost*	*DAD mechanism*

This thesis has, therefore, taken as its material for closest study the stateful protocols, as they hold out the best hope of achieving unique IP addresses, which are crucial if MANETs are to work well. To explain the main differences among stateful protocols, a new special classification for protocols following the stateful category is presented in this thesis; the details are given in the following section.

2.3 Proposed Classification of Stateful Protocols

It is, as stated, possible for the uniqueness to be ensured by concept in the stateful protocols, though the cost will be a slower assignment process and a higher signaling overhead. The main focus in attempting to improve stateful protocols is, therefore, to reduce the latency of the address assigning process to a minimum with low signaling cost. Depending on which node or

nodes are responsible for selecting, allocating and assigning IP addresses to requesting nodes, the stateful protocols can be classified into centralized and distributed management approaches as shown in Figure 2-1, with each class also divided into other sub classes. This is the new element of the classification. An elucidation of the issues follows.

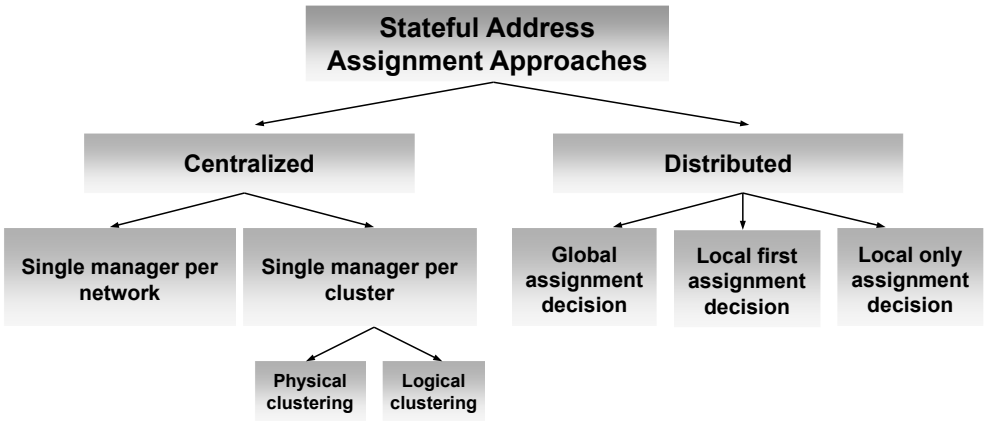


Figure 2-1: Classification of stateful address auto-configuration protocols

2.3.1 Centralized Approach

The protocols for centralized pool management basically follow the principle of the DHCP protocol by utilizing a centralized allocation table which is located in one place. This means that one node is responsible for managing other nodes. Depending on whether the network structure is clustered or not, two sub classes can be distinguished: one manager per network or one manager per cluster:

- **One manager per network:** The ABA protocol [38] which is quoted in Section 2.2.2 follows the centralized approach. Here the Address Agent (AA) is the first configured node in a network and, as with the DHCP protocol, is responsible in the whole network for managing a centralized allocation table which allows it to assign IP addresses to new joining nodes. When a node wants to join the network it is first defined as an unconnected node, then, when it obtains a unique address from AA node, as a connected node. However, if the new node is not within the transmission range of AA the other connected nodes in the network have to help in building a multi-hop connection between AA and the requester. Thus, latency in assigning an IP address depends firmly on the physical distance (number of hops) between the AA node and the requester. In a dynamic network, a crashed AA node may be a frequent event. In ABA, if a node discovers the lack of its AA node it tries to advertise itself as an AA node in the network. Advertisement contention may occur when multiple nodes discover the lack of AA at the same time; all try to announce themselves as the AA. Finally, it is a feature of ABA that an update mechanism is needed because a node must always register its IP address with the AA, and all connected nodes always have to send an acknowledg-

ment when they receive an update with new information from the AA. These two update mechanisms produce in signaling overhead by increasing number of the nodes in the network. Another example of this approach is Dynamic Address Configuration Protocol (DACP) [45] . In this protocol, the node which is responsible for assigning network addresses in a network is defined as the *Primary Address Authority* (PAA). As with the ABA protocol, the PAA node periodically broadcasts Network *Identifier Advertisement* messages to inform all nodes within the network of the network identifier. After that, each node has to register its address in PAA and it gets an address authority if there is no rejection. Basically, the Network Identifier Advertisement messages help detect possible network partitions and mergers. Specifically, when nodes do not receive their PAA advertisement for consecutive intervals, they detect the partition and elect a new PAA. The merger is detected when a different advertisement is heard by PAA which then takes the responsibility for detecting duplicate addresses in both networks. In this protocol only the nodes with the duplicate address must obtain a new address from the PAA node.

- **One manager per cluster:** to solve the problem of the increasing number of nodes in a network which may be distributed over a wide area, a network may be divided into clusters to facilitate the auto-configuration process. Basically, the management of each cluster is given to one node, which usually is called the cluster head. Similarly to "one manager per network", the cluster head has the task of allocating IP addresses to all nodes in its cluster. Moreover, it has to observe any change of the node's status in the cluster. However, to prevent any address collision with other clusters the cluster heads have to synchronize their tables with each other. In MANETs, such a synchronization process adds an additional issue to the central management issues; especially in high dynamic networks. However, the performance of protocols following this approach depends on the method used to cluster the nodes in the network, physical or logical clustering. A network is clustered physically if there is a relation between the position of a cluster head and the position of the nodes which belong to this cluster head; otherwise it is the case of logical clustering. Examples, DHAPM (Dynamic Host Auto-configuration Protocol for MANETs) [46] has physical clustering and CoReS (Configuration and Registration Scheme) [47] protocols has logical clustering. Here is an explanation of two types.
    - **Logical clustering:** CoReS protocol presented in [47] and [48] is an example of an auto-configuration protocol with Logical clustering. In this protocol a logical cluster is defined by a cluster head which is called the CR-node and cluster nodes which are the children (CH nodes) of this CR-node as shown in Figure 2-2. Basically, CR-nodes are responsible for assigning free IP addresses to joining nodes in the network. Good candidates for CR-nodes are selected depending on two values; degree of mobility (*dm*) and resource value (*R*). The IP address space is divided into many disjoint sub blocks. These blocks are distributed among the CR-nodes. In CoReS the basic idea is that a new node has to search for any neighboring CR-nodes which can directly assign IP addresses

to it. However, in many scenarios there may not be a CR-node within the transmission range of the new node (some hops away). To solve this problem, a broker node, which may be any neighboring Child node, is used to help in the address assignment. The job of the broker is to search for its default CR-node (the node which has assigned the IP address to it). Then the CR-node is responsible for assigning a free address, if it has one, to the new node. Due to the mobility in MANETs the broker may be many hops away from its default CR-node, so that undesirable Latency and Overhead is added to the assigning of IP addresses to the new nodes. Moreover, if a broker doesn't succeed in getting a response from its CR, it must first solve the abrupt departure of its default CR-node before it finds anther CR-node. Here, the joining request message of the new node will be discarded without any reply being given to the new node. Finally, to function properly this protocol has to include one synchronization mechanism among all CR-nodes and another mechanism between each CR-node and its children. Similar problems can be seen in other auto-configuration protocols which use logical clustering, such as the DHAPM protocol [46]. The main differences among these protocols are the criteria for selecting the cluster head.

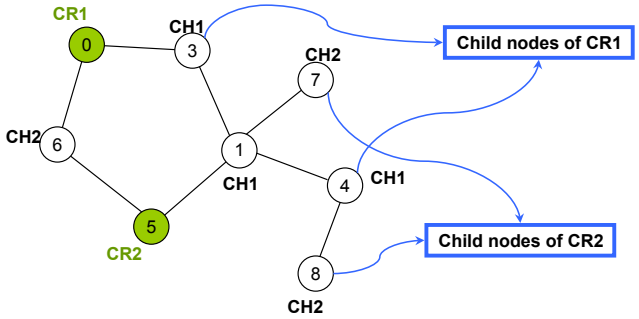


Figure 2-2: Two kinds of the nodes in CoReS protocols

In the DHAPM protocol a candidate as cluster head is selected on the basis of available (free) capacity of four metrics; memory, battery, CPU, and network interface. Basically, the increasing number of nodes in a network leads the DHAPM protocol to construct a new cluster head (i.e. the ratio of all nodes/all AAs in the MANET). The cluster head of DHAPM is called address agent AA because it manages the addresses of other normal nodes. As with CoReS, if a new joining node is not within the transmission range of an AA node, it selects a neighboring node as an initiator node. Mainly, the initiator is responsible for finding its AA node (the AA node which has assigned the IP address to this initiator). As explained above, because MANETs are by nature dynamic, the AA node may be more than two hops away from the initiator and/or it may depart abruptly from the network. To find the responsible AA, the initiator would, therefore, need a long time (assigning latency) and additional messages (signaling overhead). The maintenance process in DHAPM,

like that in CoReS, needs two kinds of synchronization: first between the AA node and its assigned nodes, second among all AA nodes in the network.

- **Physical clustering:** unlike logical clustering, the main factor defining a physical cluster in a network is the distance between the cluster head and other nodes belonging to the cluster. Usually, this distance is defined by the number of hops between each node and its cluster head. In [49] the authors introduce the Quorum protocol, in which the maximum distance in a cluster between a node and its cluster head is three hops. In this protocol a cluster head should maintain two kinds of data structure; *IPSpace* and *QuorumSpace*. The first one represents the IP address block from which this cluster head is able to assign addresses, while the second one represents the IP address blocks maintained by adjacent cluster heads. If a new node wants to join a network it has to search for the nearest cluster head by listening to the periodic hello message sent by neighboring nodes. Depending on the neighbor information the node can define the location of the cluster head. After that it has to connect with the cluster head which is responsible for assigning to it a free address. To ensure the uniqueness of an address the cluster head has to verify whether the address is occupied by checking with adjacent cluster heads in a quorum set (*QuorumSpace)*. If the proposed address is free (not allocated by other cluster heads) the cluster head assigns it to the new node. Such a mechanism adds additional latency to the assignment latency needed for the connection between the cluster head and the new node in case of more than one hop distance. The authors assume that the connection among adjacent cluster must be available for the allocation process but in a scenario of high node mobility the allocation of free addresses may lead to address conflicts because of unreliable connections. Finally this protocol suffers from high signaling overhead to ensure the physical clustering which is another big issue in MANETs.

  In [50] another auto-configuration protocol called CBA (Cluster-Based Address auto-configuration) following physical clustering is presented. Differently from Quorum, a network in CBA is clustered in 2 hierarchal levels. The authors in [50] regard an entire network as one cluster with a cluster head. Subsequently, this cluster is divided into several sub-clusters and each one of them is managed by a sub-cluster head. The cluster head manages the sub-cluster heads and assigns a prefix of address space to each sub-cluster. In each sub-cluster and similarly to Quorum, a sub-cluster head is responsible for managing all nodes located around it within a distance of three hops (maximum). Basically, when a new node wants to join the network it waits for the nearest sub-cluster head advertisement which is sent periodically. Depending on the prefix in the message and the new node ID, a candidate address is selected by the new node. Finally this address is sent to the sub-cluster head which is responsible for checking the availability of the address and selecting an available address if there is a conflict. An issue arises here if a node from one cluster moves to another cluster, in that it has to change its address because the prefix in the new

cluster is different. This is not a realistic solution, especially in high mobility scenarios. Moreover, the abrupt departure of a cluster head or any sub-cluster head adds signaling overhead because the other nodes in the network have to negotiate with each other to elect the missing head. Usually, physical clustering is more suitable for stable networks such as sensor networks in fields or forests.

2.3.2 Distributed Approach

Protocols following the distributed approach enable each node in a network to maintain an allocation table from which a configured node can select an available free address for any new requesting node. However, these protocols differ in the permission scope given to those nodes during the assignment process. The options for assignment decisions are *Global*, *Local first* or *Local only*. The concept of a *Local only* decision is that the decision for allocating and assigning a unique address to a new node is taken by any allocating node itself; while in the *Global* decision the allocating node has to ask for the assignment permission from other nodes in the network. Basically, *Local only* assignment decision approaches assume the availability of free addresses in each configured node at any time; however, in MANETs this is unrealistic because the address space is limited. Therefore, the concept of *Local first* decision represents a reasonable mechanism in which the address assignment decision is taken by the allocating node as long it owns free addresses, otherwise it searches for another node which owns free address and is ready to assign it. The following explains each approach with examples.

- **Global assignment decision:** MANETconf protocol [43] represents an example of the mechanism here, by which a common allocation table containing the available address space is distributed to all nodes in a network. However, a node cannot assign any free address selected from its table until it has received permission from all nodes in the network. In MANETconf as presented in Section 2.2.2, any neighboring node (initiator) of a requesting node can select a free address from its allocation table but to assign this address the initiator floods to all nodes in a network a request message asking for the assignment permission of the address. This means that the allocation process claims cooperation from all configured nodes in the network. If there is a conflict with any node, the initiator will receive a negative reply from this node. Then it selects another address and another attempt is initiated. Until the process succeeds, a long assigning time (latency) and control messages (overhead) must be added. Also, this approach needs a synchronization procedure to ensure that the allocation tables of the nodes are always up to date.
- **Local first assignment decision:** In this approach, the block of IP addresses is divided into disjoint blocks and distributed among configured nodes in a network. This means that every configured node, if it has free addresses in its block, has full permission to assign one of them directly to a requester node without asking for any assignment permission from other nodes in the network. As mentioned above, if the node

has no addresses free it can search for an available one among other nodes in the network. In addition to the block of free addresses, each node, in this approach, has to save in a separate table the states of all allocated addresses in the network. In such a mechanism, every node in a network is able to manage the addresses of joining or releasing nodes in the network. Mainly, most protocols from this method, such as [51], [52], [53], [54] and [55], follow the principle of the Buddy system of memory management [56] [57] to split the address space into blocks. Usually, they are called Buddy protocols. However, the Buddy protocols differ slightly from each other in the way they handle issues such as message losses, node crashes or address reuse. In this thesis the protocol presented in [52] is selected as a representative of Buddy protocols because its assignment process requires less signaling cost than protocols in [51] and [53], and it is not based on any routing protocols as are those in [54] and [55] which depend on the Optimized Link State Routing (OLSR) [58] routing protocol. Moreover, it shows an even distribution of free addresses among nodes because a new node should select an agent node which owns more free addresses. Furthermore, its function for searching for free addresses obviates a direct search in the whole network. In addition, it solves critical issues such as message losses, abrupt node departure and network partitioning/merging. Finally, the functions and algorithms of the address autoconfiguration are well described in this protocol. Basically, the assignment process in this protocol uses a handshaking mechanism between two neighboring nodes; a requester (the node requesting an IP address) and an allocator (the node that actually assigns the IP address). In MANETs a requester may have many neighboring nodes and the process starts by sending a broadcast request (*IPAddressRequest*) from the requester. Upon the receipt of this message every neighboring node checks its free IP address (*free_IP*) set and responds by sending a corresponding reply message; i.e. it sends an *IPAddressAvail* message if it has free addresses, otherwise it sends the NULL *IPAddressAvail* message. After collecting all messages sent by the neighboring nodes, the *requester* selects its allocator based on all received *free_IP* sets as follows:

- If there are non-empty *free_IP* sets, the *requester* selects the neighbor with largest block of free addresses as its *allocator*. In this case the *requester* sends an *AllocatorChosen* message with the selected allocator's IP address to all its neighbors. Upon the receipt of this message by the non-selected neighbors they know that their blocks are available to be used in another assignment process. Thus, it is ensured that nodes not chosen as the *allocator* get back their offered address blocks. In contrast, the selected *allocator* has to mark its offered block as assigned and it sends *IPAddressAssign* message with the proposed block to the *requester*. When the requester receives this message it assigns to itself the first address in the block and defines the remaining addresses as its *free_IP* set.
- Otherwise, the *requester* selects randomly as its *initiator* one of the neighbors which has no free block. This case triggers a process called *expanding ring* search in which the *initiator* has to search in other nodes for free addresses, by sending an *IPAddressInfoRequest* message to all neighbors hop by hop. It searches first among the nodes of its one hop connections. If it finds free ad-

dresses it stops the search; otherwise it increases the search by one hop and repeats the process until it finds free addresses. When the *allocator* finds free addresses it continues with the configuration of the *requester*.

In this protocol there may possibly be no free addresses in the network because all addresses have been assigned or some nodes have left the MANET without releasing their IP address and/or free_ip set. To solve the leak of IP address space due to the graceful departure of some nodes, the allocator in the expanding ring search must reestablish the presence of allocated addresses by using a "reclamation of IP address" process when it finds no free addresses. After this, it determines the addresses of the nodes that failed to respond during the expanding ring search. This means that every node in the network has to respond the allocator performing the expanding search, which results, of course, in signaling overhead especially when many allocators try to perform this process simultaneously. Moreover, most messages between the allocator and the other nodes in this process are unicast messages, which may not be suitable for high mobility scenarios.

- **Local only assignment decision:** as with *Local first* approaches, each node owns a block of the available free addresses. Basically, by using a special function in *Local only*, the node is able to define its free IP addresses. In contrast to *Local first* approaches, the node here is responsible only for the information of its free address block. This means that there is no need to save the state of other blocks managed by other nodes in a network. Of course, this can save on node resources such as memory space; however, it leads to low capability of IP management functions such as the reuse of released addresses and the detection of abrupt departures.

  A Prophet protocol [59] is an example of such mechanism, where only a local connection among neighboring nodes is needed during the assignment process. In principle, the Prophet function enables a configured node to calculate a set of unique addresses by using a seed and its IP address. So, if a new node wants to join a network, it sends a request message over one hop to its neighboring nodes and waits for a response from any of them. Basically, each configured node from the neighborhood which receives this message replies by sending a configure message and updates its own state accordingly. Depending on the information contained in the reply message, the new node configures itself with a unique IP address, initial state value, and NID which is a variable selected by the first node in the network. Because no acknowledgement is needed from the new node, Prophet suffers from possible exhausting of IP address space and (when a node reassigns the allocated addresses) consequent collisions. This means that it is only suitable for short live of networks which utilize IPv6 because of its huge address range. The Prophet problem, however, increases dramatically in scenarios when a new node sends more than one request message. This may happen in case of unreliable connections or in the presence of spoofing attacks. Therefore, in [60], the author improves the Prophet protocol by adding an acknowledgment message telling other nodes to keep any of their available addresses that are not selected by every requester.

Finally, because Prophet has no table of configured nodes it has difficulty handling issues such as abruptly departing nodes and partitioning/ merging networks.

Similarly to Prophet, the authors in [61] suppose that a new node is able to get a free address from one of its neighboring nodes. In MANETs this is not feasible due to limitation of the address space which in this protocol is divided and distributed among configured nodes. This means that in some nodes there may be no free address space. Here, the authors fail to provide a solution for the case when only one neighbor without free addresses is a neighbor for the new joining node. Moreover, this protocol fails to cope well with the abrupt departure of any father of a child node because the detection of that forces the child node to release its address and rejoin the network. This means that the node must search again for a new father. The obvious result is increased signaling cost and possible interruption of the ongoing communication in the network.

For each category of stateful auto-configuration approaches mentioned above, the address assignment features are compared in Table 2-2.

Table 2-2: Analysis of stateful auto-configuration approaches

Approaches / ***Features***	***Single manager per network***	***Single manager per cluster***		***Global assignment decision***	***Local first assignment decision***	***Local only assignment decision***
Mechanisms	-	*Logical clustering*	*Physical clustering*	-		
Average signaling cost & assignment latency	*HIGH*	*HIGH/ MEDIUM*	*MEDIUM*	*Very HIGH*	*LOW/ MEDIUM*	*Very LOW*
Possible detection of abrupt departing nodes	*YES*	*YES*	*YES*	*YES*	*YES*	*NO*
Coupling degree between address uniqueness and update process	*HIGH*	*MEDIUM*	*HIGH*	*Very HIGH*	*LOW*	*NULL*
Periodical Synchronisation process	*YES*	*YES*	*YES*	*NO*	*NO*	*NO*
Robust against Packets loss	*HIGH*	*MEDIUM*	*HIGH*	*LOW*	*MEDIUM*	*HIGH*
Robust against Nodes loss	*LOW*	*MEDIUM*	*LOW*	*HIGH*	*HIGH*	*HIGH*
Scalability	*Very LOW*	*LOW*	*MEDIUM*	*LOW*	*HIGH*	*Very HIGH*

As the table makes clear, the local assignment decision approaches outperform other approaches in most assignment features, especially signaling cost and assignment latency. This is because the whole assignment process is achieved mostly between a new node and one of its neighboring nodes without a need for periodic synchronization among nodes, additional signaling overhead is absent. However, such synchronization in turn does make the centralized approaches more robust against packet losses. The down side of centralized approaches is that they may not cope with the lack or loss of the central node which is responsible for all assignment process, and are thus unsuitable for scenarios with high speed mobility nodes.

Because *local only* approaches do not make use of a table including up-to-date information for every configured node in the network they are more suitable for scalable networks. They are, however, poor at handling some issues such as the detection and the address reuse of abruptly departing nodes. The table shows that the *local first* assignment decision approaches are able to solve this problem but they have to pay by higher cost than that in the *local only* approaches.

2.4 Conflict Resolution Mechanisms

The address conflicts referred to in this section differ from conflicts arising during the assignment process of new nodes. Here, it is a question of when all nodes in a network have been assigned unique addresses and the address conflicts have arisen due to the partitioning /merging of networks, as illustrated in Figure 2-3. Efficient detection and resolution of such conflicts is essential: an address maintenance function is crucial in auto-configuration protocols. These methods are divided into the individual and collective types, depending on their way of defining address conflicts and changing the addresses when the networks merge, as presented in Figure 2-4.

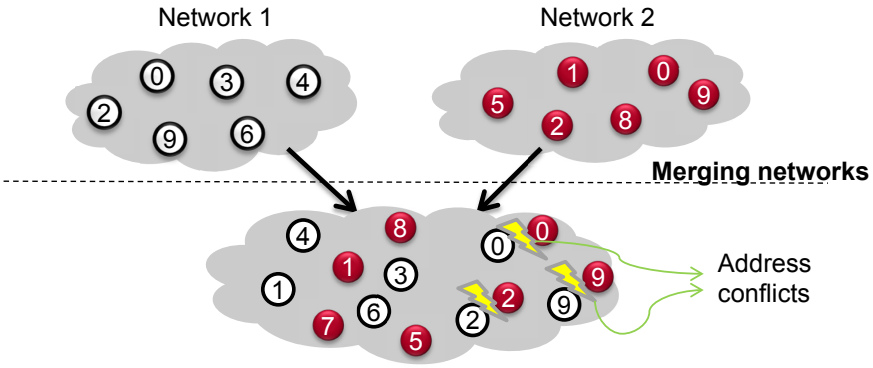


Figure 2-3: Address conflicts due to the merger of two networks

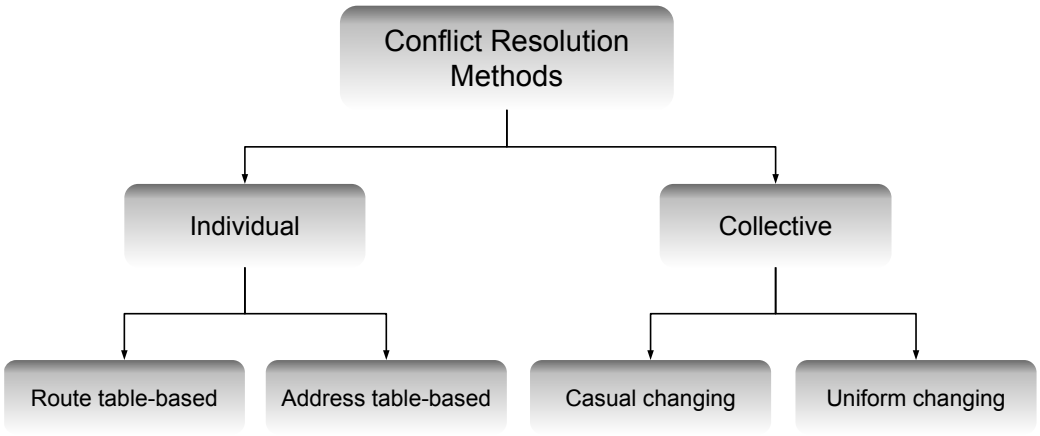


Figure 2-4: Resolution mechanisms of address conflicts in case of merging networks

2.4.1 Individual methods

The main principle here is to change only the addresses of conflicted nodes in any merged network. In the individual method, each address conflict detected by the node which is responsible for handling a merger should be solved individually. Basically, this node may be the border node which has detected the merger, or another predefined node in the network such the manager node in protocols that adopt the centralized approach. In both cases, the responsible node has to search in its network for every node with a conflicting address, then, either assigns new addresses to each of them or tells each of these nodes to search for new addresses by themselves. Therefore, the main function in this method is how the nodes of conflicting addresses are to be defined. Usually, the node information maintained in address tables or routing tables is used to permit a comparison between any received address data and the information in those tables. The conflicting nodes can then be detected and the conflict resolved. Depending on the kind of table used, two approaches can be defined: address table-based and route table-based.

- **Route table-based:** here the nodes of conflicting addresses can be determined by a check on the information in routing tables utilized in routing protocols. This means that auto-configuration protocols following this approach must couple with the routing protocols utilized in the network. Usually the auto-configuration protocols following this approach are built to work with certain routing protocols (proactive and/or reactive routing protocols). For example, the auto-configuration protocols presented in [36] , [62] and [63] depend on Optimized Link State Routing (OLSR) [58] which is a proactive routing protocol, while the protocols presented in [64] and [34] are working on the Ad Hoc on Demand Distance Vector (AODV) [65] protocol which is a reactive routing protocol. In some cases, the auto-configuration protocols may support more than one routing protocol such as the WDAD protocol [35] which can work with both OLSR and Dynamic Source Routing (DSR) [66] protocols, or PACMAN which supports OLSR, AODV and Fisheye State Routing (FSR) [67] protocols. Basically, when there are two identical addresses with different Network IDs, those auto-configuration

protocols can detect possible conflict by analyzing route messages used by the ongoing routing protocols. Then, to solve the conflict each of the conflicting nodes is informed to change its address. Although this is apparently the optimal approach because only the conflicting nodes have to change their addresses, there is high additional communication overhead to find and communicate with those conflicting nodes so that the decision can be made on which nodes have to release their addresses. Moreover, this approach results in additional delay within the routing process due to the processing time that a node has to spend checking every routing packet passing through this node. Nevertheless, the main disadvantage of any auto-configuration protocol following this approach is the hard coupling to a particular routing protocol. In MANETs this is not desirable, because of the need to modify or change routing protocols. Moreover, there is no guarantee that the approach will be at all applicable in MANETs due to the wide variety of routing protocols which have been developed for critical scenarios.

- **Address table-based:** if a node detecting a merger with another network has information (i.e. an address table) for all configured nodes in its network it can define the nodes of conflicting addresses and may be able to define the way of changing their addresses. Otherwise, the detecting node has to search for another node which has the task of handling the merger in the network because it owns an updated table of all nodes. This is the case with the protocols in [38] and [46]. Of course, the first method (if the node has up-to-date information) is faster with less overhead because there is no need to search for another node. The information of address table may be up-to-date before the node detects the merger (which is usually the case); otherwise, such information must be collected by the node during the merger. The two cases are, then:
    - If the address table is up-to-date before the merger, the conflicting addresses in merging networks can be detected directly. This happens in the MANETconf protocol [39] which allows the border nodes in the merger to detect any conflicts by exchanging their address tables which are, by the concept of this protocol, up-to-date. After doing a comparison, these nodes can select the conflicting nodes which must release their address and search for new ones. There are criteria for the selection of the releasing nodes such as the conflicting node with fewer and/or short-lived TCP connections, which would be the one to acquire a different IP address. Because this protocol does not depend on any other protocol it will be able to work in any network. However, it suffers from signaling overhead: the conflicting nodes have to search by themselves for new addresses, and, every node obtaining new address has to inform all nodes in the network about the new addresses so that the address table is kept up to date. The Filter-based Addressing Protocol (FAP) protocol [68] is similar to MANETconf in that it needs the current set of allocated addresses. Basically, the information of all allocated addresses is stored compactly by using a filter. In this protocol, when border nodes detect a merger they exchange their filters. The number of conflicted address in each network can be detected from a

comparison of the filters. Therefore, each border node floods its network with "Partition" message including the filter used by the other network. Upon receipt of this message, the conflicting nodes in a small network have to release their addresses and start a new allocation process. Depending on both filters the node randomly chooses an available address in both filters and floods the network with an AREQ message to allocate the new address. If there is a conflict with the selected address the node initiates the process for another address. If there are many conflicted addresses in the small network, the protocol suffers from signaling overhead. Moreover, the simultaneous assignment of different nodes makes address duplications highly probable because correct assignment in this protocol depends mainly on reliable flooding of AREQ messages. To decrease the duplication probability, each AREQ message is sent N_f times – again a cause of additional signaling cost.

- Otherwise, if the address table is not up-to-date, each border node detecting a merger has to search in its network for up-to-date information of all configured nodes. After that, it can define the conflicting addresses and assign to them new addresses as is the case in the Buddy protocol [52]. Because each node in Buddy is able to assign addresses to new nodes, the address table may not be up-to-date. Thus, the node detecting a merger asks other nodes in its network about assigned addresses. In this mechanism the node floods a query message to collect the network configuration information in terms of IP addresses assigned to the nodes and their respective free IP sets within its network. By aggregating all the information obtained from the response messages sent by all configured nodes, the node exchanges its table with the table of other border node which belongs to the other merging network. After comparing both tables, the node detects the conflicting addresses and the free addresses in the network. Based on this information, it performs a network-wide broadcast of a message which contains the new addresses of all invalidated addresses. After that it has to redistribute all free IP addresses among all the nodes in the merged network by sending an update message. Mainly, this protocol suffers also from high signaling overheads during the first phase (that of collecting the information of other nodes in the network). Moreover, if the information of the new addresses is not received by the conflicting nodes which may be many hops further from the sender, these conflicting nodes must repeat the merger process which adds protocol overhead.

2.4.2 Collective methods

In contrast with those following the individual method, the protocols using the collective method solve the conflicts by changing all the addresses in one of the merging networks. The mechanism is that one of the merging networks has to tell its nodes to release their addresses in order to get new addresses. This method is divided into two approaches according to the way the

new addresses are acquired - "casual or uniform" changing as depicted in the Figure 2-4.The first phase in the two approaches is identical: the node detecting the merger with another big network has to send a broadcast message in its network. Upon receipt of this message node activity varies according to the approach:

- **Casual changing:** this is when all nodes in one of merging networks have to change their addresses and search for new addresses by themselves. Usually, each node from the small merging network has to initiate a new address assignment process. This may, of course, lead in high signaling cost. Moreover, there is no guarantee that the nodes in this network will find a neighboring node possessing free addresses. High latency results, because every node need to search in further hops for free addresses. The Prophet protocol [59] is an example of such a case. In it, all nodes in a small network (i.e. one with a small number of nodes), after the merger with a big network, has to release their addresses and to obtain new ones from the large network. Similarly to Prophet, the Quorum protocol in [49] forces the nodes in a merging network with few nodes to release their addresses and to obtain new ones from the large network. Another example of casual changing of all addresses in one of the merging networks is presented in [61]. In this protocol the nodes form a tree topology, named the address tree. Basically the root node in this tree is responsible for any merger decision. This means that any node from a network receiving from a neighboring node an advertisement (Adv) packet including a different MANET ID has to inform its root node by sending a Detection_merge packet. The root (first detector) in this case should communicate with the other root node which belongs to another merger network. After a successful connection, the other root should change the addresses of all nodes in its network according to the available free addresses sent by the first detector root. The precise details are that each node in the network will get a new address from its father node wherein the root is the father for k child nodes and each child is a father for other k nodes. This mechanism suffers from high latency and signaling cost when the child nodes are further than one hop from their father. Moreover, unreliable multi-hop connection may lead to additional issues such as the detection of new merger among the nodes of same network. Finally, this mechanism depends mainly on periodic update between each pair (father, child), which in turn adds additional signaling overhead to the network.
- **Uniform changing**: in this approach the broadcast message includes information on new addresses of all nodes in one of the merging networks. This means that each node in one network will know its new address when a border node detecting a merger broadcasts a merger message. The efficient way to achieve such change seamlessly is by making a uniform change to old addresses, e.g. by modifying (i.e. increasing or decreasing) a part of all node IP addresses in one network, thus "uniformly" changing the addresses. Of course, such a uniform approach helps minimize the number of messages needed to change all IP addresses in a network because a node has no need to search for a new address when it receives the merger message instructing it to change its IP address. In addition, this approach supports other functions in MANETs

such as the routing function. Basically, a route in a routing table becomes invalid if there is any change in IP addresses representing the route entry. If a discovery process is required for the updating of each route in a routing table, such changes lead to a high signaling cost. The key advantage in the uniform approach is that an address reconfiguration does not invalidate any cached routes because new changes of IP addresses in each route are known by each node and can be easily modified. Nevertheless, finding a good means of changing all addresses uniformly is the main issue in protocols following this approach. For example, the authors in [48] present the CoReS protocol in which uniform changing is done by adding offset to all node addresses in one network (the smaller one). However, the solution shows low efficiency because only the cluster head nodes (CR-nodes) are responsible for detecting and solving the duplication resulting from the merger of the two networks, i.e. if only non-cluster nodes (child nodes) from two different networks are within transmission range of each other, they will not be able to detect the merger. In this case each node assumes abnormal cases and sends *Warn* message to its default CR-node. In the CoReS protocol, if a CR-node receives a *Warn* message or detects a merger with another network two kinds of messages are to be sent through the combined network. First the CR-nodes of the two networks have to synchronize with each other, by exchanging the information of their tables and parameters. This step is very important to help define the possible address conflicts and select the proper offset by which the nodes belonging to a conflicting rage should modify their addresses. Then, every CR-node that has to change its address range is responsible for informing its child nodes about the required changes. This process takes a long time and increases the signaling overhead in a network. Moreover, the success of the process requires reliable communication among the CR-nodes and, also, between each CR-node and its Child nodes. Finally, CoReS has no limitation on the number of bits for an address because it can expand address size as a network grows. This in turn renders the solution incompatible with the IP architecture which supports a fixed length of 32 or 128 bits.

Table 2-3 shows the main features of conflict resolution mechanisms used in MANETs. In this table we can see that uniform changing of address conflicts is preferable to other mechanisms because its average signaling cost and reassignment latency is low. Moreover, it requires no coupling with ongoing routing protocols and it allows a node detecting a merger to assign new addresses to conflicting nodes. Finally, because there is no need for mutual exchanges of signals among conflicting nodes to decide which of them is to be changed, the algorithm of this mechanism is low in complexity. However, most auto-configuration protocols which utilize this mechanism are stateful and belong to the centralized approaches which are unsuitable for many scenarios in dynamic ad hoc networks. Therefore, there is a need to find a mechanism which allows distributed protocols that follow this mechanism.

Table 2-3: The features of conflict resolution approaches in MANETs

Approaches / *Features*	***Route table-based***	***Address table-based***	***Casual changing***	***Uniform changing***
Average signaling cost & reassignment latency	*MEDIUM*	*MEDIUM / HIGH*	*Very HIGH*	*LOW*
Possible address reassignment by the node detecting the conflicts	*NO*	*YES*	*NO*	*YES*
Coupling with routing protocol	*YES*	*NO*	*NO*	*NO*
Algorithm Complexity	*HIGH*	*MEDIUM*	*LOW*	*LOW*

2.5 Network Partitioning Detection

In MANETs, network partitioning takes place and if all partitions keep working independently there will be no critical problem (i.e. no resulting address conflicts). However, if later those partitions merge together again, the problem of conflicted addresses may arise. This problem can be clarified in two cases as follows:

- The first case arises if two nodes from two independent partitions allocate identical free addresses to new nodes. This may happen when every node has permission to allocate a free address from a table containing all available addresses in the network, as in the MANETconf protocol. Here, if two nodes from different partitions allocate identical addresses at the same time there will be no rejection from the other nodes in either partition.
- The second case is when, before the merger with the other partitions, a partition reassigns addresses which have been regarded as abrupt departure nodes because they belong to the other unreachable partitions.

The detection of such conflicts may be difficult because each partition has identical network configuration (network ID). Therefore, partitioning detection is required so that each partition can be assigned with unique identifier. This will enable them to be regarded as different networks, distinguished by their unique identifiers. In this case a merger can be detected and the merger conflicts can then be handled by using one of the methods described in the above section.

Basically, there would be no assignment of identical free addresses in different partitions in protocols following *local first* or *local only* approaches because every node in those protocols owns a disjoint block of free addresses. However, if there is a need to reuse the addresses of ab-

ruptly departing nodes (nodes not found, i.e. missing node) due to the lack of available addresses, the detection of the network partitioning is important.

Because the detection of the partitioning may be a complex and bandwidth consuming process some authors depend on the tables of routing protocols to detect the departure of nodes, as described in [54]. In MANETs, the dependency on the routing table information is inappropriate because there is such a wide variety of routing protocols. Aside from the routing service, the partitioning detection can be achieved in a proactive or reactive means, depending on the impact of the partitioning on the assignment process of each protocol.

- **Proactive detection:** protocols which utilize synchronization among nodes such as centralized protocols (*one manager per network* class or *one manager per cluster* class) can proactively detect any partitioning in the network. After the detection, each partition has to change its partition identifier and select the manager in the partition (if not already found). In this way, each partition can resume the allocation of free addresses or reallocate the addresses which are of nodes detected as missing. Of course, the detection of a partitioning increases the signaling cost in the network. Moreover, if the partitions merge again later, another change needs to be done. Therefore, the protocols using proactive detection will perform poorly in networks with highly frequent partitioning and merging.
- **Reactive detection:** in this mechanism, applied by most protocols following the distributed approach, partitioning in a network will be detected when there is an event such as the assignment of free address in the network. In reactive detection there are two basic methods, reflecting the relation between the assignment process and the kind of assigned address (an address not assigned before or an address of a missing node):
    - *During the assignment of any free address:* the protocols following this method require an acknowledgment in every assignment process from all configured nodes, similarly to protocols from the *global assignment decision* class. Basically, the abrupt departure nodes (signifying partition) can be detected by any node assigning a free address to a requesting node. In MANETconf, for example, if there is a partitioning in the network, the nodes in every partition with no response during the assignment process are considered as abruptly departing nodes and their addresses will be cleaned out of the allocation table to permit the reuse in the future. To prevent possible collisions after the merger, the absence of the lowest IP address node in a partition is the key for changing the partition identifier. This means that only the partition with no such node will change its identifier. This, however, may cause address duplication in a scenario when only the partition with the lowest IP address node makes the address assignment process. In this case, this partition will reuse the nodes from another partition (in which the node with the lowest IP address is missing) without the need for a change of the partition identifier. If later the both partitions merge, there is no way to detect the possible address conflicts. To solve this problem, the authors of MANETconf protocol introduced an additional method (proactive) in which this node (with lowest IP address) periodically

broadcasts messages advertising its presence to all nodes in the partition. This in turn increases the signaling overhead in the network. Contrary to MANETconf, the FAP protocol uses flooding during the address assignment of every new joining node to modify the partition identifier (partition signatures), which depends on a filter representing the presence of all nodes in a network. However, in this mechanism if two partitions of a network assign identical addresses, their signature will be identical and the results will be address conflicts if the two partitions come back together. Moreover, the reuse of the abrupt departure of nodes in each partition is solved by flooding an AREQ message N_f times by every node when the filter in the partition is full. This overcharges the network in terms of control load because all nodes have to send this message.

- *During the reassignment of a missing address*: in a network, if a configured node fails to assign a free address from its block, or from any block of other nodes in the network, to any new node, it tries to reuse the addresses of the missing nodes, as in protocols from the *local first decision* class. The partitioning in this case can be detected when the node asks about the existence of the other configured nodes in the network, as described in the Buddy protocol. If the node detects the absence of many nodes it supposes there is a partition in the network. In this case, the addresses of missing nodes can be reallocated and the network identifier in this partition should be changed to avoid any address conflict if the partitions merge later. Such mechanism is, of course, more efficient and suitable for scenarios with highly frequent partitioning and merging networks.

2.6 Scenario-based Comparison

For the purposes of a comparative analysis of protocols which follow stateful approaches, representative protocols from each class of the classification proposed in Section 2.3 are here selected. Representing the centralized approach is ABA (one manager per network) together with two "one manager per cluster" protocols, CAB with its physical and CoReS with its logical method. MANETconf is used to represent the distributed approach in respect of global assignment decision methods, and for the *local only* and the *local first* assignment decision the representatives are Prophet and Buddy respectively.

The focus is on these representatives because in their classes they show better performance as to signaling cost (number of messages) required in the address assignment, independence of routing protocols and efficiency in their partitioning/merging mechanisms (as presented in the above sections). For example, Buddy is selected from its class of protocols because it needs low signaling cost (a minimum of 4 signals between new node and its neighbors) and it utilizes a good mechanism to solve issues such as node departure partitioning/merging without any dependence on routing protocols. Because the assignment process in centralized approaches is very similar, the CoReS and CAB protocols, which utilize a good merging mechanism, are the ones selected.

Prophet is a natural choice because it has the fastest addressing protocol among all stateful protocols. So, for the comparison of the selected protocols four scenarios are now defined, representing the main aspects of the nature of MANETs. The four scenarios vary as to density, mobility, scalability and heterogeneity, see below. Figure 2-5 is a schematic representation of the suitability of the stateful protocols.

- **Density (nodes per unit size):** the main factor here is the number of nodes per unit or transmission range. Usually, increasing node density is the main reason for high collisions and contentions in a network. The assignment process of protocols with high signaling cost such as MANETconf protocol will be particularly sensitive to this factor. In addition, protocols such as ABA and CBA may suffer in this case if multiple nodes join the network simultaneously. This is because one allocator is responsible in an area crowded with nodes. Protocols such as Buddy, Prophet and CoReS may work better because more than one node may be responsible for assigning new address. However, Prophet outperforms other protocols because the assignment process needs less signaling cost than all other protocols.
- **Mobility (average nodes speed):** here it is assumed that the nodes move inside the network without causing network partition. This means that the direct connection between two nodes may be interrupted due to the mobility but indirect connection will always be possible via intermediate nodes between the two communicating nodes. In such case centralized protocols which utilize synchronization mechanism suffer greatly because the connection between an allocator and other nodes belonging to this allocator is not always possible. Basically, such protocols add high signaling cost to ensure uniqueness among nodes in high mobility scenarios. Finally, Prophet is more suitable than the other protocols because its local assignment process has less signaling cost.
- **Scalability (area size of distribution area):** the main factor describing scalability in this scenario is the number of hops between the two furthest nodes in the network. This means that the nodes in the network are distributed over a large area, in which many hops are constructed. Such a case affects mainly the assignment process in which allocator nodes may be many hops further from requesting nodes. Scalability is worst addressed by the ABA and CoReS protocols because in them an allocator in many cases may be many hops further from a new node requesting an address. Of course MANETconf, too, is less than adequate because the decision of an address allocation is done globally.
- **Heterogeneity (partitions & merging networks):** when a network frequently partitions and/or merges with other networks, the issues of duplicate assignments and resolving of address conflicts may arise. The case here may be difficult when information of all nodes in the network is not available – this applies to the Prophet protocol which follows *local only* method. In this protocol all nodes in a network must search by themselves to change their addresses after each merger. At the same time, when periodic synchronization for up-to-date information is needed, the frequent por-

tioning and merging add high signaling overhead for protocols such as ABA and CAB to solve missing nodes and avoid duplicate assignment later.

In the figure, it is obvious that none of those protocols are suitable for all the scenarios studied. For example Prophet works very well in the first three scenarios because it does not need information concerning other nodes in a network, but for just that reason it has drawbacks in merging and partitioning scenarios. Therefore, there is a need to design an efficient protocol which may work properly in all the different scenarios, as shown in the figure by the orange dashed line. It is with the aim of meeting this need that this thesis presents the proposed Logical Hierarchical Addressing (LHA) protocol which will be described in the next chapter.

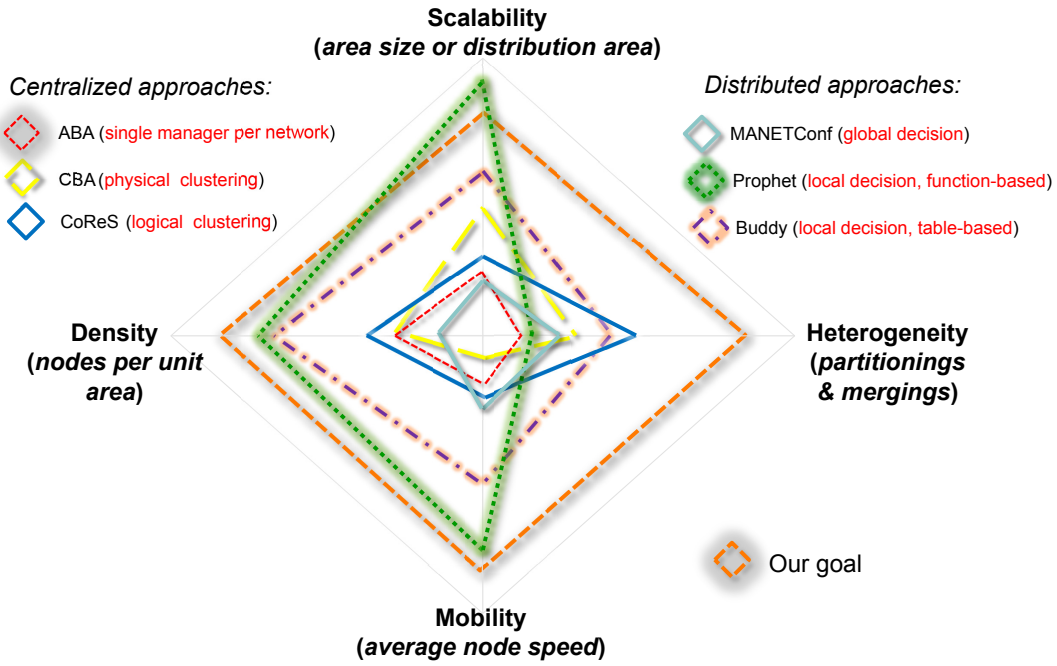


Figure 2-5: Four-dimension comparison of representative stateful protocols

Chapter 3 LHA Protocol

This Chapter is a detailed description of the new Logical Hierarchical Addressing (LHA) protocol. LHA is a stateful auto-configuration protocol applying a distributed approach with *local first* assignment decision. The protocol utilizes a new function to distribute the available address space among all nodes in a network. By means of this function, every node in the network knows the available space of other nodes. This in turn supports the recovery function in case of address space leak due to the frequent joining/departure of nodes. To enable reassignment of allocated addresses, LHA makes use of certain lists which will be described later in this chapter. Because LHA divides the Host ID (HID)[1] of IP address structure logically it can efficiently handle issues such as the partitioning and merging of ad hoc networks.

The sections are organized as follows. Section 3.1 describes the required features on which LHA protocol is designed. The basic idea of the LHA protocol is in Section3.2, and tion 3.2 3.3 presents the address assignment function designed for LHA. Section 3.4 gives an overview of the data structure required in LHA. Then the main LHA algorithms, those of node joining, network merger, network partition and node departure, are presented in detail in Sections 3.5 to 3.8.

3.1 Required Features of LHA

LHA must efficiently handle the address assignment and maintenance tasks for every possible event in a network. In MANETs, a distinction is made between the events which may emanate from a single node and from a group of nodes (referred as a network or a portion). While the events of a single node are known as joining and departure of a node, the events of a group of nodes (network) are described as merger and partition of networks. Figure 3-1 represents the functions/features which are required in LHA to efficiently handle each of those events, and which dictate the design of the protocol.

[1] In IPv4 address HID could be referred to sometimes host number as used in [94] or host/local address as used in [91].

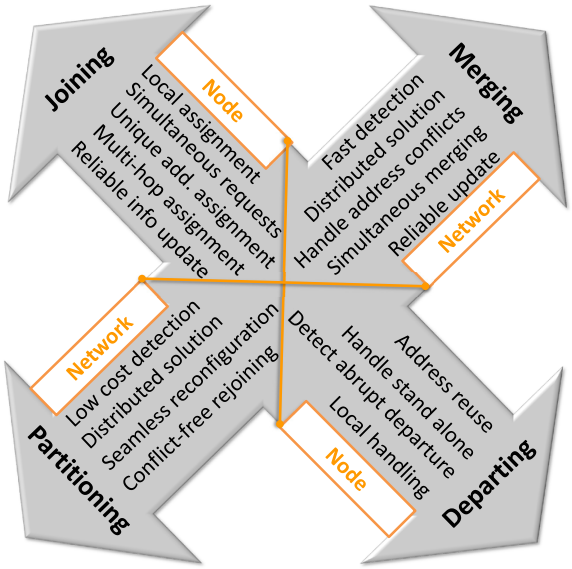


Figure 3-1: LHA required function/features

- **Joining of nodes**
  - *Local address assignment:* to avoid high assignment latency and reduce the signaling cost.
  - *Handle simultaneous requests:* to serve different nodes joining the network at the same time a configured node which may get two requests from different nodes should be able to serve them.
  - *Unique address assignment:* to avoid address conflicts in the network.
  - *Multi-hop address assignment:* to avoid a lack of free addresses at the neighboring node of a joining node, which could arise because the number of free addresses at each node is limited.
  - *Reliable update:* to provide every node with an overview of the number of all nodes in the network, which in turn enables accurate decision in every possible scenario, e.g. reallocating of missing addresses in network partition scenarios.
- **Departure of nodes**
  - *Local Handling:* to enable a fast departure process by ensuring that it is the neighboring nodes of a departing node, which are responsible for handling the departure request from this node.
  - *Address Reuse:* to avoid a lack of address space during frequent joining/departure of nodes and to ensure that the reused addresses do not cause later any conflict in the network.

- *Handle alone standing node:* to avoid inaccurate data update when a node stands alone (no connection to another nodes in the network).
- *Detect abrupt departure (missing nodes):* to enable the reassignment of the lost address or addresses when there is a need for that.

- **Partitioning of network**
  - *Low cost detection:* to avoid the high signaling cost due to periodical announcement, the detection of network partition should take place on request.
  - *Distributed solution*: to make the solution robust against possible communication faults with decreased detection time, each node in the network should be able to detect and handle any partitioning.
  - *Seamless reconfiguration:* after the detection, a new configuration may be needed in each partition. To avoid unnecessary communication overhead and a large number of table changes, there is a need for seamless way of reconfiguring the partition nodes.
  - *Conflict-free rejoining:* in one partition it would in principle be possible for all nodes of other partitions to be detected as missing nodes. Therefore, the reuse of IP addresses in one partition may lead to address conflicts if the partitions reunite after some time has elapsed. In this case there is a need for a good method of preventing such address conflicts.

- **Merger of networks**
  - *Fast detection:* Because there may be numerous address conflicts when networks merge, the task of detecting the merger should be done as fast as possible.
  - *Handle address conflicts:* After the detection, the hard task required is to find the conflicts and resolve them. Robust completion of this task is vital to the reliability of the protocol.
  - *Distributed solution:* to cope with many and varied merger scenarios, every node in a network needs to be able to detect and solve possible network mergers.
  - *Simultaneous merging:* one of the difficult merger issues is how to handle the merging of more than two networks at a same time. Here there is a need for an efficient mechanism which ensures a correct reconfiguration plan for the merging networks.
  - *Reliable update:* Because the mergers involve much updating in the network there is a need for a reliable update function to ensure a failure-free merger process.

3.2 Basic Idea

In this thesis, the basic configuration principle of LHA, which is applied, is that each node is able to manage a disjoint part of the available IP address space. This means that each node is able to select, allocate, and assign unique addresses from its own block without getting permission from other nodes in the network. Because LHA handles only the available space of IP addresses, it is applicable to any block of IPv4 or IPv6 addresses. However to simplify and further illustrate the assumptions underlying the present work, a block[1] 192.168.0.0 - 192.168.255.255 (192.168/16 prefix) of private IPv4 addresses [69] is used to address the hosts (nodes) within a MANET. This means that in a private IPv4 address the available address size is 16 bits for each node address (i.e. Host ID). For the efficient management of the available address space and to solve issues such as partitioning and merging, LHA imposes the logical structure of the IP blocks by dividing the Host ID into two parts, Hierarchical ID (HierID) and Hierarchical Host ID (HHID) as shown in Figure 3-2. To explain how LHA deals with both portions, the next sections show the functions of each of them. Basically, the selection of each portion size depends on an assumption that in a network the occurrence rate of address assignment events (joining/departing nodes) compared to the occurrence rate of the events of merging/partitioning networks is bigger, wherein, this ratio is about 3:2. Thus, the size of the HHID must be bigger and is calculated on the functions presented in (3.1 & 3.2). For example, if the value of HostID (16 bits) in private IPv4 block (192.168/16 prefix) is applied to (3.1 & 3.2) LHA will define 6 bits for the HierID part and 10 bits for the HHID.

Figure 3-2: IP address block in LHA protocol

In LHA the selection of unique addresses by the assigning nodes is achieved by utilizing an allocation function (see Section 3.3.1). The main advantage of using this function is the even distribution of available address blocks among configured nodes. Furthermore, it enables each node to know the address blocks of other nodes and to identify the allocator of every address in the network, which in turn helps in handling issues such as departing or missing nodes.

$$HHID_{bits} = floor\ (2 * HostID_{bits}) /3) \quad (3.1)$$

$$HierID_{bits} = HostID_{bits} - HHID_{bits} \quad (3.2)$$

[1] The private blocks of IP addresses are allocated by Internet Assigned Numbers Authority (IANA) which can offer the private IP blokes from the A, B and C classes

3.2.1 Hierarchical Host ID (HHID)

In LHA each node in a network belongs to a logical address hierarchy. The creation of an address hierarchy is managed by the first node in each address hierarchy by selecting a unique number from the range of HierID block. This means that all nodes from one hierarchy share the same HierID number. Basically, LHA utilizes the HHID block to distinguish among all nodes belonging to an address hierarchy; where it assigns a unique number, the HHID, to each node.

It is the basic idea in LHA that such an assignment process is achieved by letting the protocol treats the HHID block of an address hierarchy as a set of disjoint sub blocks, wherein each configured node belonging to this address hierarchy manages its sub block autonomously without permission from other nodes in the network. In other words, any configured node is able to act as an Address Agent (AA) which is responsible for serving a unique address to a new node (requester). The AA node assigns one of its free addresses if it has any. Otherwise, it has to search for a free one in the network. Basically, a requester has to select one of its neighboring nodes to become its AA node. If no neighbors exist, the node supposes that it is the first node in the network. In this case it has to create its address hierarchy by selecting a random number from HierID block (see following section for more details about the HierID function). After that, it selects the value of its HHID randomly from the range [0-15] and is therefore called the 'root node ' and its HHID is called '*ROOT*" parameter. In addition, the root has to define a hierarchical Network Identity (hNetID) parameter which may be its MAC address or a global unique identifier number of 16 octets selected by way of Universally Unique Identifiers (UUIDs) [70]. LHA utilizes these two parameters with HierID to identify each address hierarchy.

In the LHA approach, there are two additional roles for a network node: that of the predecessor and that of the successor. The node, which provides an address to a requesting node (requester) from among its free addresses, is called the 'predecessor' for this requester. Analogously, the requester is called the 'successor' of the node from which it has acquired a unique address. According to this relation LHA defines a number called the hierarchical level (*Hl)* which indicates the level of a node in its address hierarchy, wherein, the *Hl* value of a node increases the *Hl* of its predecessor by 1 and the *Hl* of the root node is set to 0. Finally, every predecessor can have a number (M) of direct successors, whereas every node will own only one predecessor. In this case, a logical address hierarchy is built in the network as shown in Figure 3-3.

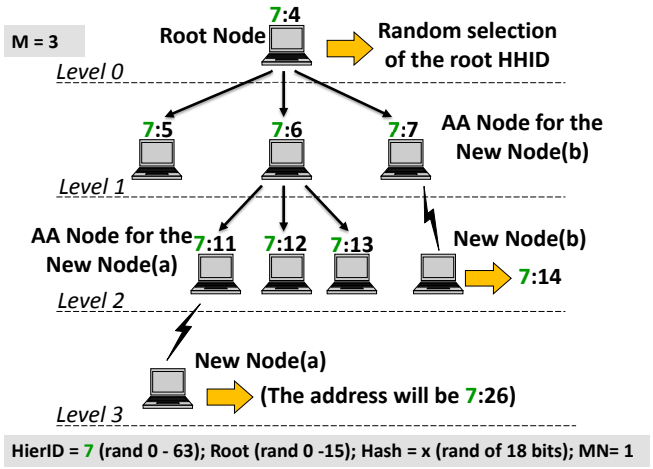


Figure 3-3: Example of the logical address hierarchy in LHA

In this figure the example is an address hierarchy where M, the maximum number of free addresses per node, is set to 3. This means that a configured node managing an address block can assign up to 3 requesters with unique IP addresses. As shown, the *Hl* of a node is defined by the node location in the address hierarchy where *Hl* of the root is set to 0. In the figure the arrow indicates the logical connection from a predecessor to its successor. However, the lighting symbol in the figure indicates a physical connection between two nodes and is used to explain the configuration process of the new joining nodes. As mentioned above, for every node in a hierarchy, the root node selects the HierID which is a part of every IP address of these nodes; in this example it selects 7 randomly from a range [0-63]. Notice that the Hash and MN parameters are utilized for the merger aspects, see next section, and are initially set by the root as shown in the figure. In addition, only the root node selects its HHID (*ROOT*) randomly from a range of [0-15]; in the figure the value is selected as 4. The HHID values of other nodes are, in contrast, calculated on the basis of the assigning Equation (3.3) discussed in Section 3.3.1 . To explain this in the figure it is assumed that the new nodes (node "a" and node "b") are located within the transmission ranges of nodes (7:11 & 7:7) respectively. Because a new node has to select one of its neighboring nodes as an Address Agent AA node, it will be the task of nodes (7:11 & 7:7) to assign a unique address to the requesting nodes. In the example, the HHIDs of new nodes (a & b) are selected in accordance with the Equation (3.1) as 7:26 and 7:14 respectively. Finally, every node has to save the LHA parameters presented in Table 3-1.

Table 3-1: LHA assignment parameters used by a node

Parameters	*Description*
Hl	*The level of the node in the hierarchy* *where* $Hl_{root} = 0$
Av	*The current available number of free addresses at this node* *where* $(0 \leq Av \leq M)$
CN	*The current number of all configured nodes* *where* $(0 \leq CN \leq 2^{HHID(bits\ size)})$
Seq	*The sequence number of the next available address in the assigning list* *where* $(0 \leq Seg \leq M)$
LN	*The current number of lost nodes* *where* $(0 \leq LN \leq 2^{HHID(bits\ size)})$
M	*The maximum number of free addresses that a configured node may own* *where* $(0 \leq M \leq 15)$
ROOT	*The HHID of the root node in the hierarchy* *where* $(0 \leq ROOT \leq 15)$

3.2.2 Hierarchical ID (HierID)

Using the HierID, LHA permits the creation of more than one logical address hierarchy in a MANET as illustrated in Figure 3-4 which shows an example of a network including two address hierarchies. This may happen if there is no possibility of assigning free addresses to requesters because, for instance, that all available addresses have already been assigned. For example, suppose that at first only the address hierarchy of $HierID_x$ and $ROOT_x$ exists in the network given in the figure. So, if a new node joins the network and there are no free addresses in the address hierarchy of $HierID_x$, the AA node of the joining node will instruct it to construct a new address hierarchy (y). Following the LHA algorithm, the $HierID_y$ and $hNetID_y$ parameters are selected as described in Section 3.2.1 and the new node is defined as the root node $ROOT_y$. After that, the new root informs other nodes (from x address hierarchy) in the network about the configuration of the new address hierarchy. Finally, each node in the network has to increase the index of Merger Networks (MN) by one and save the information of both hierarchies in a table called the

merging list (see Table 3-5). Thus, if any other new node joins the network it gets a free address from the new root $ROOT_y$ and it will then be a part of the address hierarchy of $HierID_y$.

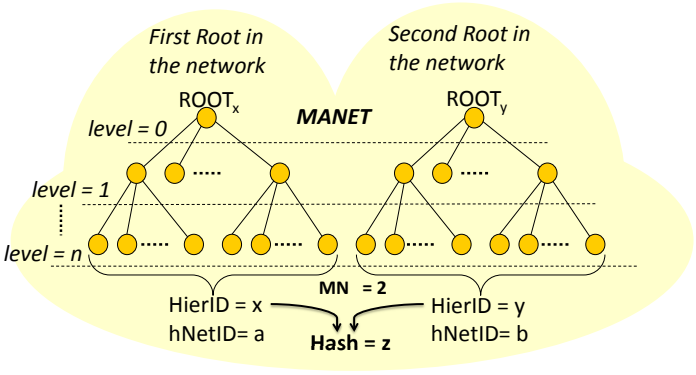


Figure 3-4: An example of two address hierarchies in a network

Because a network may include more than one address hierarchy, there is a need to assign to this network a specific identity number which will distinguish this network from other networks with identical HierIDs. This identity number helps mainly to solve merging issues when the merging networks include one or more identical HierIDs. So to discover such cases, every node in a network utilizes a new parameter called a "Hash" which is a hash value calculated from all HierIDs saved in the merging table. The example in the figure shows this parameter where the inputs of the hash function are the HierIDs (x and y) and the output is the 'Hash' value (z). Because each node in a network can calculate the 'Hash' value from its table, LHA supports a distributed function for handling network partitions and mergers as will be covered later in Section 3.6 and 3.7.

3.3 Address Assignment

This section introduces the assigning function of LHA, which distributes the available address space (i.e. HHID block) evenly across all nodes belonging to one address hierarchy and enables a configured node to select and allocate from its sub block a unique IP address to a requester.

3.3.1 LHA Function

Let hA_r, hA_{ag} and hA_n be abbreviations referring to "hierarchal address (a number from HHID block) of "root", "agent" and "new node" respectively. Let hA_{ag}^{i} and seq^{i} be the address of the agent that can offer an available IP address to node *i* and the sequence number of available addresses at the agent of node *i* respectively. Let *M* be the maximum number of available address at any node in the networks. The LHA function that calculates the new address hA_n^i of a node *i* can be written as follows.

$$hA_n^i = (1 - M)hA_r + M * hA_{ag}^i + seq^i \ , \quad 1 \le seq \le M \tag{3.3}$$

3.3.2 Correctness of IP Address Assignment

- **Given:** Prior to an address assignment all the nodes have unique IP addresses.
- **Assertion:** Following the IP address assignment of LHA function all the nodes have unique IP addresses and disjoint free IP sets.
- **Proof:** the assertion is proved when two different nodes assign IP addresses to new joining nodes. Here it will be shown that each node is able to calculate a unique address to a requesting node by using the LHA equation without having address conflicts with any address that may be calculated by another node.

Here is a case to clarify the proof: two new nodes are joining a network as in Figure 3-5. Following equation (3.3), the new addresses for node *i* and *j* are given as (3.4) and (3.5) respectively:

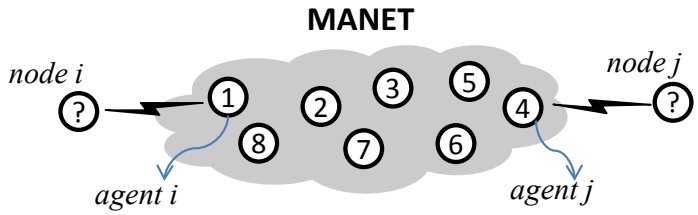


Figure 3-5: Assumption of address assignment for two new joining nodes

$$hA_n^i = (1 - M)hA_r + M * hA_{ag}^i + seq^i \tag{3.4}$$

$$hA_n^j = (1 - M)hA_r + M * hA_{ag}^j + seq^j \tag{3.5}$$

What is to be proved is that $hA_n^i \neq hA_n^j$ when the new addresses are assigned by two different agent nodes, say hA_{ag}^i and hA_{ag}^j as shown in the figure, where $hA_{ag}^i \neq hA_{ag}^j$. Since the difference d between the IP addresses of pair (i, j) in the network will be in the range 1 to $K-1$ where K is the maximum value of an address allowed for a node to use in the network, then the following relations are true:

$$hA_{ag}^i = hA_{ag}^j + d, \quad 1 \le d \le K-1, \quad 0 \le i,j \le K, \quad i \neq j, \quad hA_{ag}^i > hA_{ag}^j \tag{3.6}$$

and

$$hA_{ag}^j = hA_{ag}^i + d, \quad 1 \le d \le K-1, \quad 0 \le i,j \le K, \quad i \neq j, \quad hA_{ag}^i < hA_{ag}^j \tag{3.7}$$

Let $hA_{ag}^{i} > hA_{ag}^{j}$ and hA_{ag}^{i} be substituted from (3.6) into (3.4). This yields

$$hA_{n}^{i} = (1 - M)hA_{r} + M(hA_{ag}^{j} + d) + seq^{i}$$

$$hA_{n}^{i} = (1 - M)hA_{r} + M * hA_{ag}^{j} + (M * d) + seq^{i} \quad (3.8)$$

Using (3.5) in (3.8) gives

$$hA_{n}^{i} = (1 - M)hA_{r} + hA_{n}^{j} - (1 - M)hA_{r} - seq^{j} + (M * d) + seq^{i}$$

$$hA_{n}^{i} = hA_{n}^{j} + \psi, \quad \psi \neq 0 \quad (3.9)$$

where $\psi = seq^{i} - seq^{j} + (M * d)$. Hence, $hA_{n}^{i} \neq hA_{n}^{j}$ and the correctness is held for $hA_{ag}^{i} \neq hA_{ag}^{j}$. The correctness for given $hA_{ag}^{i} < hA_{ag}^{j}$ can be proved in a similar manner.

3.4 Data Structures

Nodes running LHA require the maintenance of local data structures to perform LHA functions. Each node uses the following local data structures.

3.4.1 Tables & Parameters

In LHA every node maintains 6 lists. A configured nodes list enables the node to take a decision in handling the issue of missing addresses when there are no more free addresses for the assignment process in the network. If the network merges with another network, the information in a merger list is needed. To reuse the addresses of departure nodes every node, has to save the departure information in a departure nodes list. Because a configured node which has allocated all its free addresses may need to search for free addresses in the network, a reverse path list is used to enable the connection over multiple hops. To reduce signaling cost among neighboring nodes when the allocation service in some nodes is better than others, a sending list is required. Finally, each node is able to manage efficiently its available addresses because it saves the status of its available addresses in the assigning table. In the following detailed information about required data and parameters of each list is presented.

3.4.1.1 Configured nodes list

Each node maintains information about all assigned IP addresses in the address hierarchy (from the root node to the last configured node in the network). Because the unique assignment of addresses does not depend on the up-to-date information of this table, LHA does not need a periodic update of this table. Basically, the information in each configured node can be collected directly from the broadcast announcement of every joining node or gradually from the Beacon messages sent from the neighboring nodes of the node. As shown in Table 3-2, each entry is a quadruple of elements which are firstly information of IP/MAC addresses for each configured node, secondly the IP/MAC addresses of the predecessor of this node, thirdly the information of the joining time and fourthly the status (missing or available) information of this configured node in the network. As this table shows, decisions to handle issues such as missing nodes and address conflict detection can be made by each node. Moreover, a comparison of the information in this table with other node information helps in detecting network mergers and partitions.

Table 3-2: Configured nodes list

Index	*Predecessor*		*Configured node*		*Status*	*Joining time (Jt)*
	MAC	*IP*	*MAC*	*IP*		

3.4.1.2 Assigning table

Once a node is configured it builds its assigning table as illustrated in Table 3-3. This table consists of four columns and k number of rows, where k=M.

Table 3-3: Assigning table for the management task of available addresses in each node

Sequence number (Seg)	*Assign Status (S)*	*Requester MAC*	*Confirm timeout (CT)*

Each row in this assigning table represents the assignment information of the successor of this node. Therefore, the columns Requester MAC and Confirm Timeout (CT) are initially set to null. The first column (Seq) includes, in each row, a potential sequence number which can be assigned to any requester node. The second column, Assign Status (S), shows the status of the sequence numbers in each row representing status, “free” “pending” and “assigned” respectively. While “free” indicates that the sequence number in this row can be used directly in any assignment process, “pending” and “assigned” represents the unavailability. However, “pending” differs from “assigned” status in that the Seq number is only temporary unavailable (i.e. for a specific time defined in column CT). In LHA, when a node selects an available address for a requester it changes its status from “free” to “pending”, and when it sends its reply message to the requester it starts a timer for the selected Seq number wherein the CT value in the row of this Seq indicates

the waiting times until the node receives an acknowledgment from the requester node. If this timer runs out prior to receiving the acknowledgment, the node supposes that the address has not been used by the requester and it changes its status to "free". In the other case, when the node receives an acknowledgment from the requester node, it resets the timer and sets the status to "assigned" which means the Seq number is no longer available for other assignment processes.

3.4.1.3 Departure nodes list

Each node saves in this list (Table 3-4) information about every node managing the address of a departing node when this manager is not the direct predecessor of this departure. This may happen when the predecessor does not exist in the network due to a departure (normal or abrupt). In the table the current manager column refers to a responsible node of the IP address of the departure node. In LHA, if the predecessor of the departure node exists in the network there is no need to save information in this table because the predecessor modifies the information of its departing successors in its assigning table, changing the status from "assigned" to "free" as described above.

Table 3-4: Departure nodes list

Index	*Current manager IP*	*Predecessor IP*	*Departure node*			*Departing time*
			MAC	*IP*	*Seq*	

3.4.1.4 Merger list

LHA depends on this list to solve the merging problems in MANETs. An entry in this list consists of seven elements as shown in Table 3-5. Mainly, a node saves in this list information of all address hierarchies included in the merging networks wherein an entry of this list consists of the parameters *ROOT, HierID, hNetID and CN* from Table 3-1. In addition, the node saves addresses of the last and second last configured addresses, $\boldsymbol{last_{conf}}$ and $\boldsymbol{S\_last_{conf}}$.

Table 3-5: A merger list for saving a trace of all address hierarchies in a network

Index (MN)	*ROOT*	*HierID*	*hNetID*	*CN*	$last_{conf}$	S_last_{conf}

3.4.1.5 Sending list

To enable a fast response from nodes having more free addresses during the assignment process and to reduce the load[1] of messages sent or forwarded by neighboring nodes, the sending list is used. Each entry of Table 3-6 records the information of certain messages scheduled to be sent to other nodes when a specific timeout is reached.

[1] The high network load is the main factor of packet collisions in wireless networks

Table 3-6: Sending list

Sequence number	*Destination MAC (one hop)*	*Send/forward timeout*

3.4.1.6 Reverse path list

Because LHA utilizes control messages over multiple hops, this list is employed to trace every reverse path between each pair (source, destination). This mechanism is similar to the route discovery function used in routing protocols such as AODV [65]. As shown in Table 3-7, each entry includes information of the sequence ID of a message, the IP address of the initiator which is the AA node of the new node, the MAC address of the new node, the MAC address of the precursor which is a next hop intermediate node on the reverse path, the hop number to the initiator, the life time for each entry, and the route status which enables a node to reduce the number of sent messages on the reverse path from different destinations. In LHA different nodes may respond by sending positive replies but one is sufficient for a successful assignment process. Therefore, an inactive value means that there is no need to send a message on its reverse path.

Table 3-7: Reverse path list

Sequence ID	*Initiator (AA node) IP*	*New node MAC*	*Precursor MAC*	*Hop count*	*Life time*	*Route Status (active or not)*

3.4.2 Messages

LHA is designed to achieve the functions of the assignment and maintenance of unique IP addresses in MANETs. These functions, as mentioned in Section 3.1, are used to handle four main actions defined as joining/departing nodes and partitioning/merging networks. With regard to those actions, the LHA messages can be grouped into four kinds which are described in the following. In this thesis the protocol packet is presented with IPv4 addresses as depicted in Figure 3-6.

3.4.2.1 Packet Format

Figure 3-6 shows the basic layout of every LHA packet which consists of three main parts, packet header, message header and message body. However, for easy testing of LHA under Linux, as presented in [71], the LHA messages are encapsulated[1] within the packet format of the

[1] The encapsulation method for a protocol's messages is used in the designing of many protocols; such as the Dynamic RARP (DRARP) protocol [95]

ARP protocol [72]. This means that the packet header in the figure is replaced with the corresponding MAC header used by ARP. Now follows a description of each field.

- Packet Header
  - Packet Length: The length (in bytes) of the packet
  - Packet Sequence Number: it MUST be incremented by one each time a new LHA packet is transmitted. Thus, the nodes in the network can define new packets sent by every node in the network.
- Message Header
  - Message Type: a specific number is defined for every message utilized in LHA. Because of this rule, LHA functions can successfully handle all LHA messages which are presented in the following section.
  - Hash and MN parameters as described in Section 3.2.2 are important to the handling of different issues such as merging/partitioning networks.
  - Time to Life (TTL): to decrease message overhead and suit the scalability aspect LHA messages need this parameter, which is mainly defined by the maximum number of hops.
  - Message Sequence: to distinguish the message from other messages from the same type sent by any node. For example, a node may try to send another request message if the first attempt has failed. In this way other nodes know that their first responses have been not correctly received.
  - Hop Count: indicates the number of hops a message has attained. Initially, the originator of the message sets the Hop Count to "0", and before a message is retransmitted by other nodes, this must be incremented by 1. If the Hop Count is equal or bigger than TTL the message must not be retransmitted.

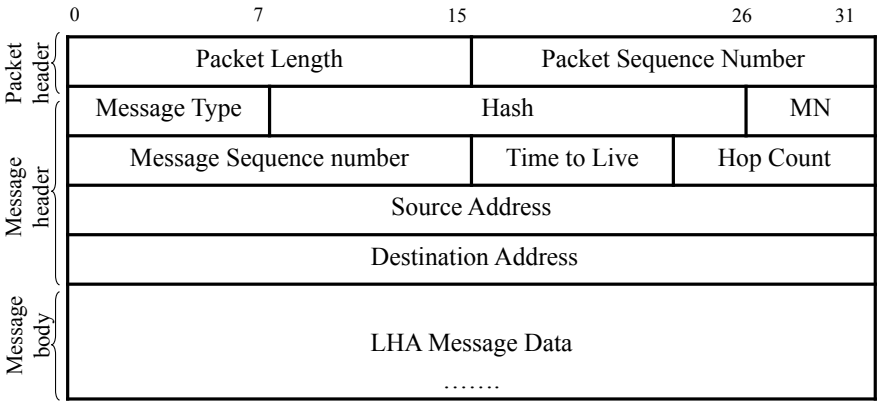


Figure 3-6: LHA message format

- Message Body: as shown in section above, LHA uses tables and parameters to archive its functions successfully. So, in this part of the message LHA can send any saved information to other nodes when there is the necessity for that. Because multiple address hierarchies are used in LHA the size of the data a node needs to send is low and in most cases one packet is sufficient to send LHA messages.

3.4.2.1 Joining nodes messages

The following messages are exchanged during IP address assignment:

- **Address Agent Solicitation (AA_Sol) message:** this message is sent by a new joining node as a local broadcast message to neighboring nodes in the network for requesting an IP address. Basically this message does not need to include any data structure; otherwise, the MN of this message must be set to "0" which indicates that this node does not belong to any address hierarchy.
- **Address Agent Reply (AA_Rep):** one-hop broadcast message sent by any neighbor node as the response to the AA_Sol message. It includes the network configuration and the possible free address provided by the sender. It comprises the following elements:
  - LHA parameters presented in Table 3-1 and hNetID.
  - If the node owns free address ($Av > 0$) it sends lists of the configured nodes, departure nodes and merger; otherwise ($Av = 0$) there is only merger list.
- **New node (New_Node):** this message is broadcast by a new node to inform all nodes in the network about its new configuration. It contains its hNetID, MAC/IP addresses, joining time and the MAC/IP address of its predecessor.
- **Address Agent Selection (AA_Sel):** when a new node does not get any offered address from all received AA_Rep messages it has to send a unicast message to select one of the responding neighbors as AA node. This message contains only the MAC/IP addresses of the selected node.
- **Address Agent Address Request (AA_A_Req):** this message is sent by any node selected as AA node which does not have addresses free for the assignment. In this case it broadcasts this message in the network to search for free addresses in other nodes. The sender must set TTL in the header and there is no need for any data structure.
- **Address Agent Address Reply (AA_A_Rep):** any node owning free an address and receiving an AA_A_Request from an AA node has to respond to the sender by sending this message including the address it is offering. The message also contains the MAC/IP and hNetID of the predecessor, the configured nodes list and the LHA parameters including the selected Seq number.

- **Address Agent Confirmation (AA_Conf):** when an AA node finishes the searching phase for free addresses, it unicasts AA_Conf to the predefined requesting node. There are two cases here:
    - There is a free address in the network: the message is sent to inform the requester about the new address, where it contains the selected Seq number, MAC/IP, hNetID of the predecessor and lists of the configured nodes, departure nodes and merger.
    - Otherwise ($Av = 0$), the message must include this parameter with the merger list. In this case the requester has to construct a new address hierarchy because there are no free addresses in the network.

3.4.2.2 Departing nodes messages

- **Departure Agent Request (DA_Req):** when a node wants to be released from the network it sends this message to its neighbors in order to select one of them as DA. This message contains the data (MAC/IP, ROOT and hNetID) about the predecessor of this node.
- **Departure Agent Reply (DA_Rep):** a node responds with this message to inform the requesting departure node that it is ready to manage its IP address. This message includes the number of free addresses (Av) of the sender.
- **Departure Agent Select (DA_Sel):** a node can safely release the network when the node sends this message to the selected neighboring node. This is because the selected node will be responsible for managing the address of the departing node. The node must then send in this message its assigning list, departure nodes list and the data (MAC/IP, ROOT and hNetID) of its predecessor.
- **Departure Management (Dep_Manage):** this message is sent by a DA to inform other nodes in the network that it is ready to manage the issues of a defined departing node and, moreover, to enquire about the existence of the predecessor of the departure node. This message must contain the assigning list and the data (MAC/IP, ROOT and hNetID) of the departure node.
- **Predecessor Existence (Pre_Ex):** DA node sends this message in the network to ensure if there is a predecessor of the departing node. This message contains identical information to the one in Dep_Manage.
- **Departure Node (Dep_Node):** this message is broadcast in the network by a node which is responsible for managing the address of a departing node. This means that the IP address of the departing node can be reallocated to any new node by the sender of this message. Mainly, Dep_Node message informs the nodes in a network to modify their data base because there is a node departing the network. This message contains hNetID, ROOT and the MAC/IP of both nodes (departure node and its predecessor).

- **Missing Node (Mis_Node):** this message is broadcast in a network when a configured node discovers that there is a missing node. In LHA the discovery of missing nodes exists in different cases; one of these cases is when a DA node does not get a response from the predecessor of a departing node. This message contains the data (MAC/IP, ROOT and hNetID) of the missing nodes.

3.4.2.3 Partitioning messages

- **New_HierID:** this message is broadcast to all nodes in a partition when a node detects the network partitioning. Upon receipt of this message each node has to change its HierID into the new one included in the message (see Section 3.7.2 for more details). Basically, this message contains the configured list and the old and new information (HierID and hNetID) of the network.
- **Existence Request (Ex_Req):** this message is broadcast by a node detecting a partitioning in its network. Its purpose is to define the number of missing nodes in the network. This message contains the data (ROOT and hNetID) and a set of addresses of the nodes which have not assigned all their free addresses in the network.
- **Existence Confirmation (Ex_Conf):** this message is sent by a node when it receives Ex_Req message (including its address in the list) to inform the requester about its existence in the network. Mainly, the message contains the LHA parameters, hNetID and the assigning table of the sender.

3.4.2.4 Merging messages

- **Information Request (Info_Req):** when a node detects a Beacon message including a different network configuration it unicasts to the sender of the Beacon Info_Req message including its LHA parameters and the lists of the merger, departure nodes and configured nodes. In a special case the message may include, in addition to these parameters and lists, a free address if the node owns a free one and if, at the same time, it detects in the Beacon message a merger with an unstable network, see Section 3.6.5. Usually, Info_Req is sent between two border nodes from different networks but it is possible for it to be sent between two nodes from a same network if one of them does not have up-to-date information, e.g. the last configured nodes in the Beacon are different.
- **Information Reply (Info_Rep):** when a node receives Info_Req message from another node it has to reply by Info_Rep message including its parameters and lists as in Info_Req.
- **Soft Merger (S_Merge):** if a node detects a soft merger between its network and another network (after comparing its merger list with another received list included in any of the info messages) the node broadcasts S_Merger to all nodes in its network to update their configuration data. This message contains the merger list of the other merging network.

- **Hard Merger (H_Merge):** this message is sent by a border node when the merger (after detection) may lead to address conflicts. This message informs all nodes to change their configuration to the new one included in the message. Basically, this message contains hNetIDs and HierIDs of the addresses hierarchies which must use the new HierIDs, and the merger list of the other merging network.
- **Release Configuration (Rel_Conf):** in the special merger case (merging with a very small network, e.g. less than 5 nodes) a border node in the small network must send to the nodes in its network Rel_Conf message to inform them to release their configuration and get new one from the big network. This message contains only the data of HierID, ROOT and hNetID of the sender.

3.4.2.5 LHA Beacon message:

Basically beaconing messages are widely deployed in MANETs [73,74] for localization and to enable rapid detection of node movements, including network splitting and merging. The protocol presented in [73] defines in each domain a *Beacon* node which is responsible for generating a Beacon message with a unique ID at a configurable rate. From this message this protocol can detect network topology changes. In order to cover the network domain, *non-Beacon* nodes within a domain must forward the Beacon message if it is not at the border of the domain.

Similar to this protocol, LHA uses the Beacon message to detect the network merging; however, contrary to the mechanism already described, each node in LHA has to send its Beacon message because LHA supports a distributed mechanism to detect and handle different issues such as the merger of networks and the update of the configured nodes list. In LHA, the Beacon message must include information about HierID, Root, CN, Jt_{last} (joining time of the last node) and the last two configured nodes ($last_{conf}$ and S_last_{conf}) in the address hierarchy of every node. In addition, the number of merged networks (MN) and the Hash number must be included in the Beacon. By means of these parameters in the Beacon message each node is able to analyze the states of its neighbors as shown in the flowchart in Figure 3-7, afterwards deciding upon a corresponding step to avoid later any possible problem in the network. Basically, this flowchart explains the LHA procedure in each node when it receives a Beacon message. For example, if a node is not standing alone (i.e. its MN is not 0, see Section 3.7.4) and it receives a Beacon with at least MN or Hash is different from its own, the node detects a possible merger. To define the kind of the merger, in this case, the detecting node must send an Inf_Req message to the neighbor that sends the Beacon (for more details see the network merger in Section 3.6). Moreover, the Beacon message in LHA is employed as an additional mechanism to update the configured nodes list as shown in the figure, wherein it provides the last updated information ($last_{conf}$ / Jt_{last} and S_last_{conf}) from the configured nodes list in each node. In this information, if the last configured node is not in the configured nodes list the detecting node has to add it and modify its parameters accordingly. If there is any conflict the node exchanges info messages with the sender of the beacon. This in turn allows the node to take a correct decision, as will be explained later in LHA algorithms, wherein LHA is defined to solve four main issues referred to as the joining/departing of a node and the partitioning/merging of a network. In the following sections, algorithms required for each action will be introduced.

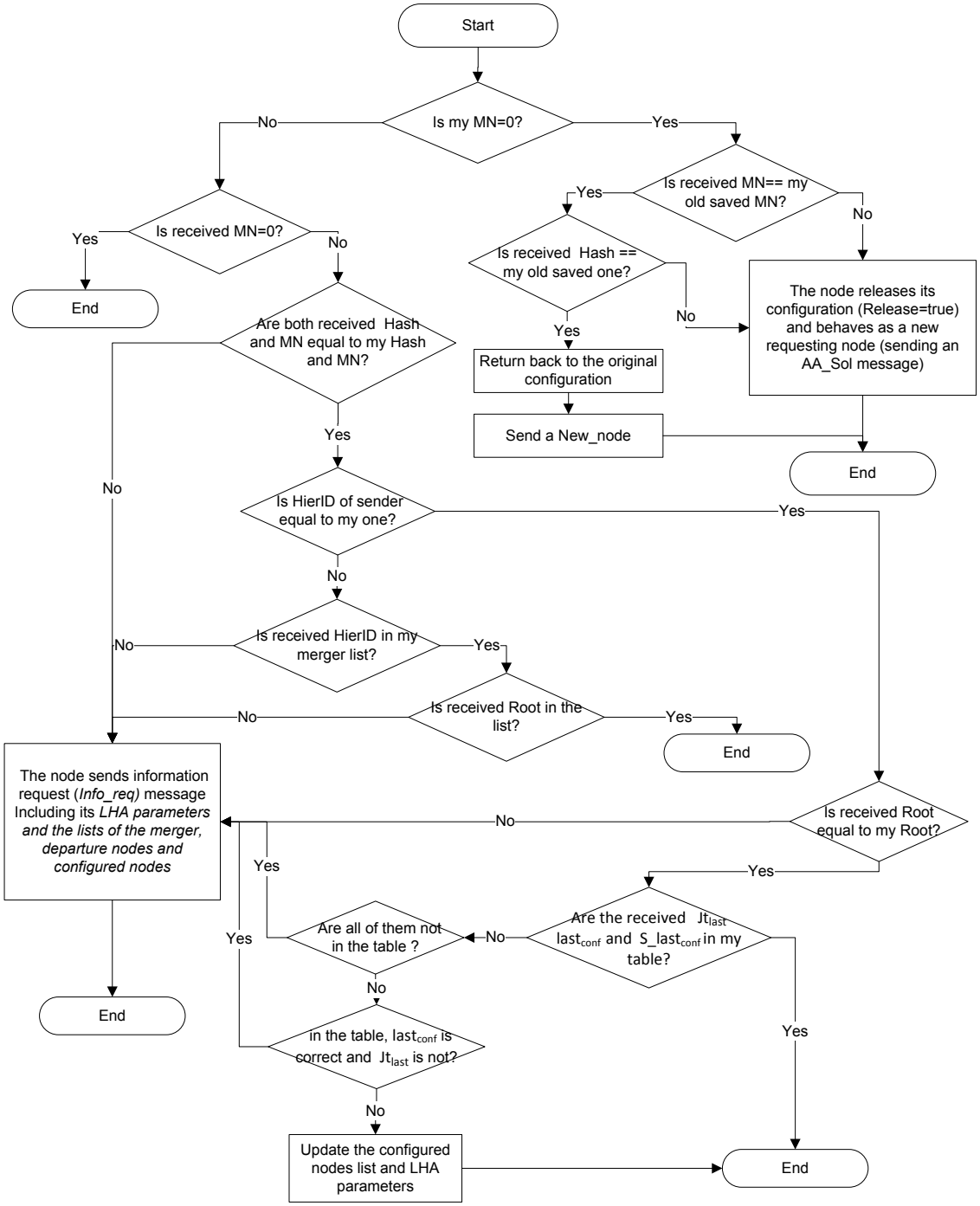


Figure 3-7: The flowchart of the Beacon message procedure for each node in LHA

3.5 Node Joining Algorithms

Usually, node joins a network in one of two separate scenarios; when (1) the node is powered on or when (2) the node connecting a network leaves its network and wants to join a new one. However, because a node in the first scenario may not find a predefined network when it is powered on, the node in this case must initialize the network by itself. Such a case is referred as network initialization (or network construction, see the following section) and is considered as a special case of node joining. In the basic case, however, a node, during the joining action, obtains the configuration of the new network, which in turn allows the node to communicate with other nodes in the network. In LHA a joining node gets a new configuration from any configured node in the network. It is assumed here that the joining node and other configured nodes are able to communicate locally by means of their MAC address or by way of Universally Unique Identifiers (UUIDs[1]) [12] as described in [11]. In the following the algorithms will be described in detail.

3.5.1 Network Initialization

In the initial (special) case, when a node is the first one in the network, it starts a solicitation timer (Tm-sol) and broadcasts a AA_Sol message. During the solicitation period the node waits to receive an Address Agent Reply (AA_Rep) message. If no replies are received after a predefined number of retries (R), the node concludes that it is the first node in the network. In this way, the node initializes a MANET by electing itself as a root node for one of the address hierarchies in the network and configures itself with the first address in the address hierarchy, setting the following parameters of its table ($Hl = 0,\ CN = 1,\ Seq = 0,\ ROOT = HHID_r,\ Av_r = M$). Details of this case are given with an example in Section 3.2.1. In addition, the root defines the identifier number of its address hierarchy hNetID which will be saved with the other parameters in the merging list (see Table 3-5). Thus each node is able to detect the possible merging and partitioning networks in the future.

3.5.2 One-hop Assigning (Basic Case)

In contrast to the initialization case, the one-hop or multi-hop assigning scenarios present the case that the new joining node is not the first node in the network. This means that there is at least one neighboring node within the transmission range of the new node. The difference between the one-hop and multi-hop scenarios is that in the one hop assigning at least one of the neighboring nodes that responds to the new node request has an available free address to be allocated to the requester while, if none of the neighboring nodes has available addresses, the new

[1] UUIDs are 16 octets long, and as they are exchanged in messages only between neighboring routers, they need only be locally unique.

node will be in the multi-hop scenario. In LHA, the *Av* parameter indicates the current number of available free addresses, here, *Av>0* indicates that the node can directly assign a unique address to the requester.

Now, the one-hop scenario can be clarified by supposing that the neighboring nodes have *Av>0*. Figure 3-8 shows the MSC[1] of a new node with two neighbors (configured nodes) where *Av>0*. So, when a new node arrives at the network the assigning algorithm of the LHA protocol executes the following steps:

- The new node (requester) broadcasts an Address Agent Solicitation (AA_Sol) message to its neighbors.
- Upon receipt of this message, each neighboring node choses from its assigning table (Table 3-3) a Seq number of status "free", changes the status to "pending" and decreases the value of *Av* parameter by 1. Before the response of an Address Agent Reply (AA_Rep) message, the node sets an allocation timeout (Tm-allo). Basically, the node sends an AA_Rep message which includes its parameters and the lists of the configured nodes, departure nodes and merging networks.
- When the new node receives an AA_Rep message including *Av*>0, it selects the sending node as its predecessor. This means that the node uses the seq value sent in the message to calculate its address and builds its tables according to the information received. Then it broadcasts a New Node (New_Node) message in the network.
- Upon receipt of the New_Node message, each node modifies its parameters and tables according to the selected new address in the network and then forwards the message to other nodes. Moreover, each node which has provided a free address to the new node must change according to the selected address the assign status of its provided Seq, Table 3-3. In this way, the nodes whose their addresses are not selected by the new node change their assign status to "free"; however, the predecessor of the new node changes the status of the selected Seq in the assigning table to "assigned".

[1] Message Sequence Charts (MSCs) [93]

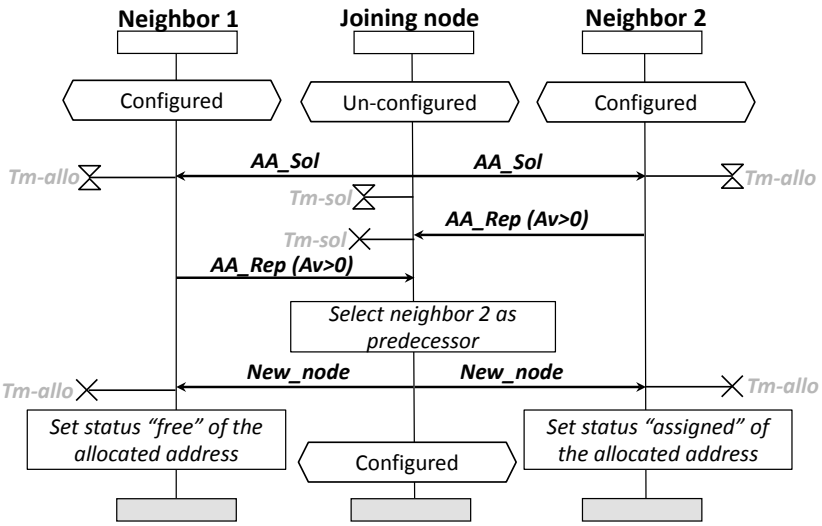


Figure 3-8: One hop assigning algorithm without wait-send timeout

Because all neighboring nodes, as shown in Figure 3-8, have to respond simultaneously, the assigning algorithm suffers from high signaling overhead, especially in dense nodes. This in turn leads to low assignment performance due to the high network load. To eliminate this and ensure a balanced distribution of available free addresses among the nodes, an additional timer with a period wait-send Timeout (Tm-ws) based on Av has been defined to be used by the neighboring nodes when they receive the AA_Sol message. The MSC diagram in Figure 3-9 shows two neighboring nodes 1 and 2 that select different values of their Tm-ws timeouts because they have different Av values, for example Av in node2 is bigger. In this assigning algorithm, each node is able to send its AA_Rep message only after the expiration of its timer. During Tm-ws timeout, e.g., the node1 may receive a New_Node message sent by the new node. If this case occurs, node1 knows that the new node has obtained a new address from another neighbor, which means there is no need to send AA_Rep message to the requester. To ensure the difference in Tm-ws, values the selection of every value is based on the Av value and HHIDs of neighboring nodes in each responding node. The following two steps explain the selection mechanism[1]:

- First, each node defines a main timeout range from equation 3.10, where RTT here is an estimated round trip time[2] over a one-hop communication in ad hoc networks.

$$main\ range = [(M - Av) * RTT \rightarrow (M - Av + 1) * RTT] \qquad 3.10$$

- Second, the node sorts the HHIDs of its own and its neighbors, and assigns each one with an index starting from 0 for the smallest one. Based on the number of all HHIDs it divides the main timeout range into sub-ranges and according to its index it assigns

[1] Similar to the mechanism utilized in [79], which ensures a low likelihood of selecting identical timeouts by the neighboring nodes that try to reply or forward the same message
[2] The time required for a signal to travel from a specific source to a specific destination and back again.

itself a corresponding sub-range. Finally it selects a random number from this sub-range as its Tm-ws.

In this way, LHA reduces the network load likelihood of multiple transmissions of AA_Rep messages, which in turn leads to high reliability of packet delivery. Moreover, it ensures a balanced distribution of available free addresses among the nodes because the node which has more free addresses will respond faster than other nodes which have fewer. Therefore, LHA in this thesis follows this algorithm which differs crucially from the other ones in prior publications describing LHA, as in [75] [76] [77] [78].

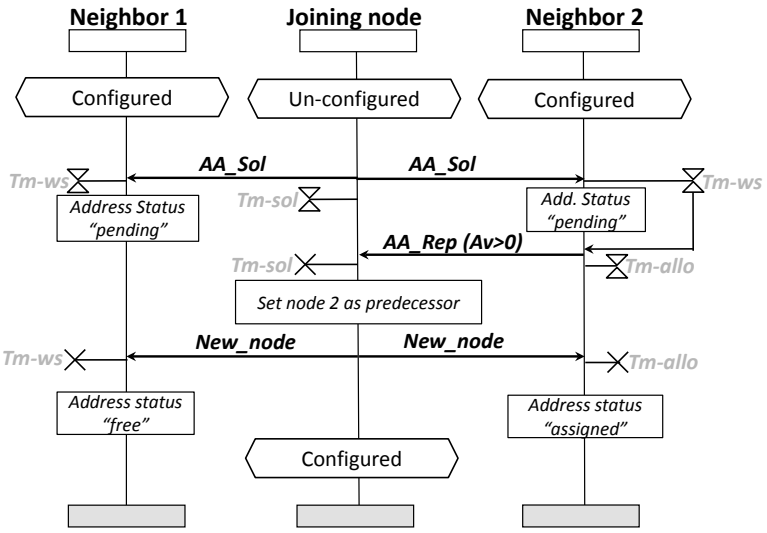


Figure 3-9: One hop assigning algorithm by using wait-send timeout

3.5.3 Multi-hop Assigning (Basic Case)

In contrast to one-hop assigning, the multi-hop assignment describes the case where none of the neighboring nodes of the joining node owns free available addresses for the allocation process. In this case, when *Av* parameters of all neighboring nodes are equal to zero, a search for free addresses beyond the neighboring nodes must be made. Because the new node, in this case, does not get a network address it is not able to communicate over multiple hops in the network. As dictated by the LHA algorithm, the searching mechanism must be achieved by one of the neighboring nodes. Therefore, the new node has to select one of its neighbors as AA to do this work. To describe the algorithm a simple scenario as illustrated in Figure 3-10 where the exchanges of AA_Sol and AA_Rep messages among nodes are done successfully is taken as an example. For this scenario the algorithm is described in steps as follows:

- After the reception of a first AA_Rep message with *Av* equal to zero, as shown in ure 3-10, the joining node sets a collecting timeout (Tm-coll). During this timeout the node collects all AA_Rep messages of (*Av=0*). After the timeout and depending on the number of neighboring nodes in every message, the node selects the last responding

neighbor to work as an AA node. Then, it sends an AA_Sel message to the selected AA, sets a configuration timeout (Tm-conf) and waits for an AA_Conf message.

- Upon receipt of the AA_Sel message by the selected AA node, it sets a search timeout (Tm-search) to limit the searching time. The node then starts the searching timer and sends an Address Agent Address Request (AA_A_Req) message to the nodes in the network.
- Each node receiving this message will check its *Av* parameter. If it has a free address, it replies with an Address Agent Reply (AA_A_Rep) message. If it does not have any free addresses, it will forward the AA_A_Req message to its neighbors. However, to avoid flooding all nodes in the network with this message, the number of hops throughout this forwarding process is defined by the sender in the field of the maximum number of hops. Moreover, the node uses Tm-ws timeout for this message and on overhearing any AA_A_Rep message from at least one of its neighboring nodes it ceases to forward the AA_A_Req message.
- During the Tm-search period, if AA node receives a reply message, it stops the timer and sends an AA_Conf message including the configuration information offered by the replying node to the new node.
- After the receipt of AA_Conf, the new node uses the address from its predecessor, which, in this case, is not the AA node. Then, it builds its tables and sends New_Node to all nodes in the network. Basically, if some nodes fail to receive this message because of the drop issue in unreliable broadcasting, this will not cause any problem during the assignment process in LHA. Contrary to other protocols with global assignment decision (such as MANETconf [43] or FAP [68] protocols), the assignment process in LHA does not depend on the up-to-dateness of information for all configured nodes in the network. Moreover, LHA utilizes another update mechanism to ensure the reception of this message by using the Beacon messages of neighboring nodes as presented in Section 3.4.2.5. Finally, because the new node must send this message only once without any need for acknowledgments from other nodes or periodical announcement of its presence, the sending of this message does not count as high overhead in the network.
- As described in one-hop assigning, upon receipt of the New_Node message, each node modifies its parameters and tables according to the new IP address in the network and forwards the message to other nodes. The message also informs the predecessor node that its free address has been used by the new node, causing predecessor to change the status of its allocated address in the assigning table to “assigned”.

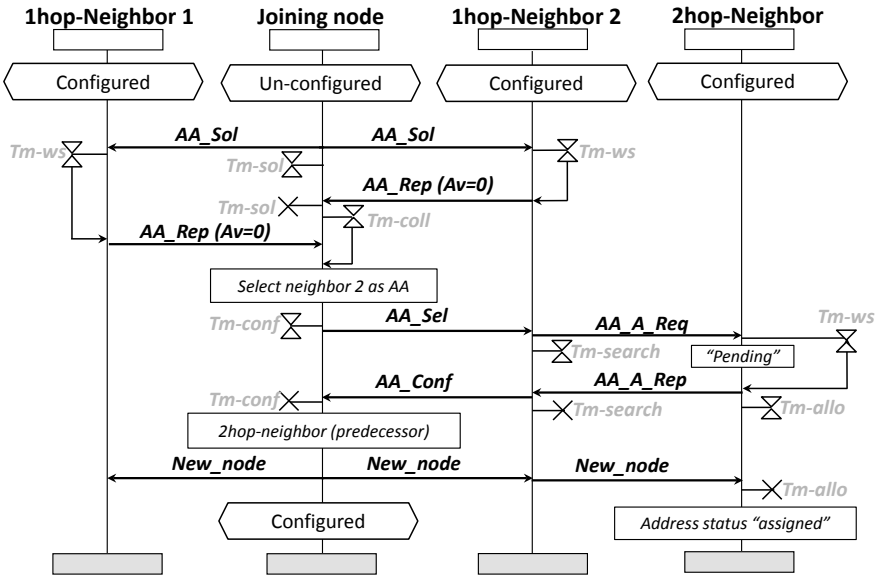


Figure 3-10: The assigning algorithm for a multi-hop scenario

3.5.4 Complete Specification of Assigning Algorithm

In this section the UML[1] state machine is used to give a whole overview of main states in the LHA protocol. However, the UML description of different states representing big algorithms which have many procedures and notations is infeasible on the state machine. Therefore, Abstract Protocol Notation (APN) is used in some states to specify the node behaviors of LHA in detail. APN is chosen because it is capable of use in UML when there is a need to describe a specific state in detail. Moreover, the description form of APN is similar to the form used in the programming languages, which in turn facilitates the implementation work later.

Figure 3-11 gives an overview of the main states of a node in the LHA protocol. In this figure the LHA protocol has three main states:

- "Un-configured": indicates the state in which a node is joining a network. In this state the new node tries to get new networking configuration from the neighboring nodes of a new network.
- "Configured (active)": indicates the status in which a node has an accepted configuration in the network and is able (active) to assign, allocate and manage available addresses in the network.
- "Wait departure (inactive)": indicates the status when a configured node is waiting for a seamless release from its network (i.e. a way to enable its address to be reused with

[1] Unified Modeling Language (UML)

no problem later). In this state the node has available configuration but is not able (inactive) to do normal actions, such as IP address assignment or maintenance.

In the figure, a state may consist of sub-states which may be sequential or parallel states as shown in "Un-configured" state or "Configured" state respectively.

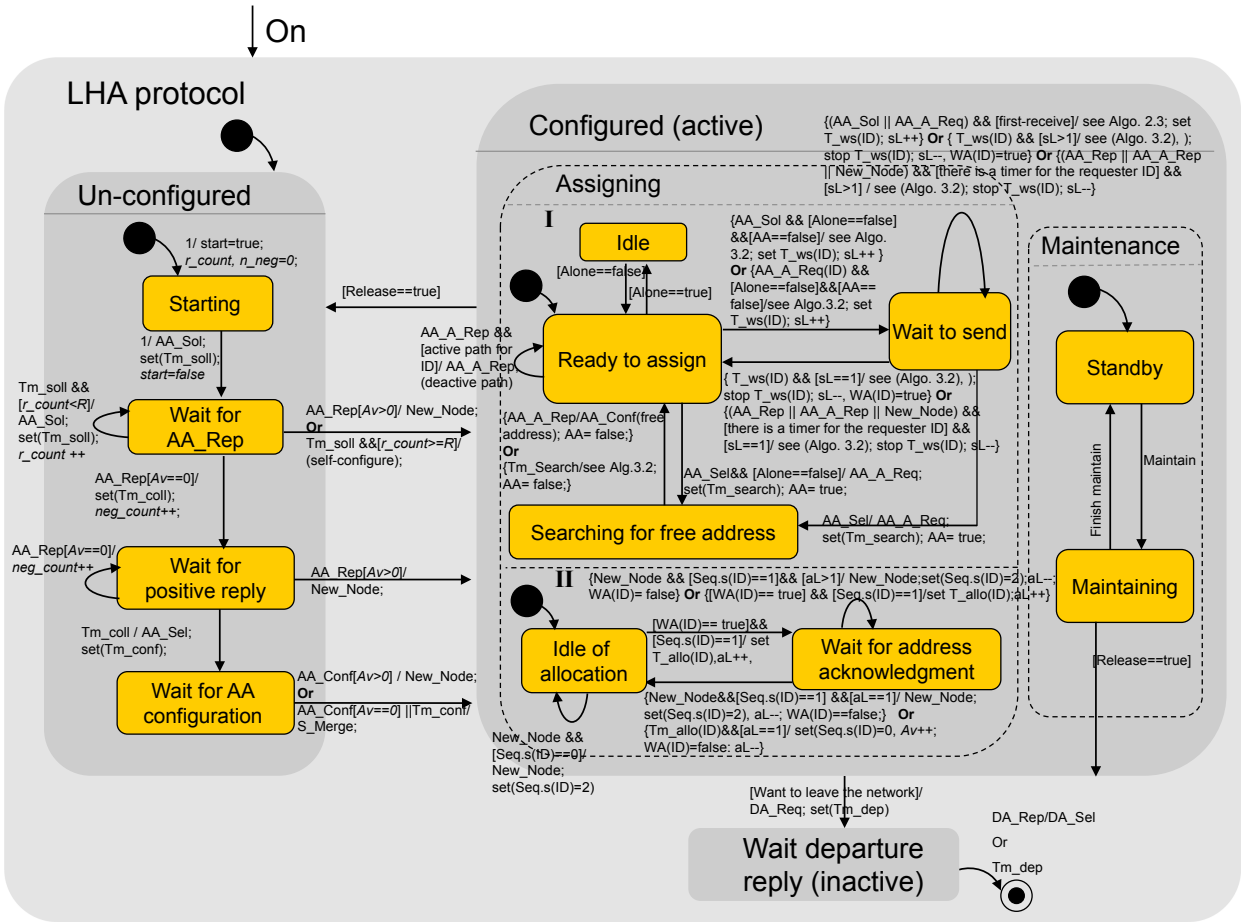


Figure 3-11: Main state[1] machine of LHA Protocol

Basically the configuration process in LHA starts from the case where an un-configured node wants to join the network, so the LHA state machine starts from the Un-configured state as follows:

- When a mobile node switches to ad-hoc mode, it enters the state "Starting". In this state the network name/mask and IP type can be selected. Also, LHA has to set initial values for its parameters as in Algorithm 3.1, e.g. it sets the values of the retry and negative counters (*r_count=0, neg_count=0*). After that, it broadcasts a request message (AA_Sol) to its neighbors. Notice that only a one-hop broadcast is necessary because the new node

[1] Main state: some utilized variables{(sL: sending list index), (aL : allocation (or assigning) table index), (Seq.s(ID): the status of the Seq number in the assigning table for a requester with this ID), (T_ws(ID) wait-send timeout for this ID), (T_allo(ID) wait-ack. Timeout for this ID), (WA wait ack. "*Boolean"*)}

has no IP address. Then, it starts the solicitation timer (*solicitationTimer*). Finally, the node changes from "Starting" state to "Wait for AA_Rep" state.

- During the "Wait for AA_Rep" state (where *neg_count=0*) the mobile node waits for the receipt of an AA_Rep message or the expiration of solicitation timer. According to these events, possible scenarios are the following cases:
    - If Tm-soll is expired (*solicitationTimer* is reached in alg. 3.1) the node increases the *r_count* by 1. Here there are two cases reflecting the *r_count* value in Algorithm 3.1 as follows:
        - If *r_count<R,* the node sets a new solicitation timer. Then it resends the AA_Sol message and stays in the waiter state.
        - If *r_count >=R*, the node knows that there are no neighbors within the transmission range and it is the first node in the network. Therefore it configures itself with a root address for an address hierarchy as described in Section 3.2.1 and enters "Configured" state.
    - If the mobile node receives a reply during the solicitation time, it checks the number of free addresses included in the message. Here, two cases can be defined as follows:
        - If $Av > 0$ (positive reply), the node knows that the neighbor has allocated a free address to it. Therefore, the node uses the allocated new IP address and depending on the received network configuration it sets the information of its tables and parameters. Then the node stops *solicitationTimer* and informs all nodes in its network by broadcasting a New_Node message. After that, it changes its status to "Configured" state.
        - If $Av = 0$ (negative reply), the node saves the information of the replying node and increases a negative counter (*neg_count*) by 1. To improve the likelihood of receiving a positive reply the node sets Tm-coll (starts *collectionTimer* in Alg. 3.1) and enters the state "positive reply".
- "Wait for positive reply" indicates that the node has received at least one negative reply (*neg_count > 0*). In this state the node waits to receive a positive reply until the *collectionTimer* is reached. During the timeout the node collects all negative replies. The node changes its state if there is one of the two following events:
    - If the node receives a positive reply it uses the configuration information included in the message and broadcasts a New_Node message to inform all nodes in its network. After that, it stops the *collectionTimer* and changes its status to "Configured" state.
    - If *collectionTimer* expires, the node ensures that there is no possibility of receiving a positive reply from the neighbors. So the node has to select one of the respondents to work as an AA node which in turn will be responsible for searching for a free address for the new node in the network. Thus, the new node sends an

AA_Sel message to the selected node. Then, it sets Tm-conf (starts *configurationTimer* in Alg. 3.1) and enters the state "Wait for AA configuration".

- If, while in the "Wait for AA configuration" state, the node gets a positive response (AA_Conf message including a new address), it configures itself with the HHID, hNetID and HierID included in the message. After the node is configured it broadcasts its new address (New_Node message) and changes to "Configured" state. Otherwise, if there is a negative response (AA_Conf with no free address) or the configuration timer expires, it chooses a different HierID from the ones received in the reply message. Then it calculates the Hash of all HierIDs including its selected one and configures itself as a new root for an address hierarchy in the network. After that it broadcasts the new configuration with the S_Merge message and changes to "Configured" state.

Algorithm 3.1: joining algorithm of Un-configured node

```
const R;                    {a threshold of retry #}
bool start := true;
bool configured := false;
int r_count := 0;           { retry counter }
int neg_count := 0;         { negative counter }

start = true →
       broadcast AA_Sol;
       start solicitationTimer;
       start := false;

receive AA_Rep from neighbor →
       if free IP is available {positive reply, Av >0} then
              if neg_count =0 then
                     stop solicitationTimer;
              else
                     stop collectionTimer;
              fi
              send New_node to all nodes;
              configured := true;
              skip;
       else
              if neg_count = 0 then
                     stop solicitationTimer;
                     start collectionTimer;
              fi
              neg_count := neg_count + 1;
       fi

receive AA_Conf from AA node →
       stop configurationTimer;
              if message includes free address then
                     send New_node to all nodes;
              else
                     self-configure();{build a new address hierarchy}
                     CalculateNewHash();{from its HierID and received ones}
```

```
                    send S_Merge to all nodes;
      configured := true;
      skip;

timeout(solicitationTimer) →
      if r_count < R then
            broadcast AA_Sol;
            start solicitationTimer;
            r_count := r_count + 1;
      else
            self-configure(); {selection of HierID, ROOT and hNetID }
            configured := true;
            skip;
      fi

timeout(collectionTimer) →
      selectAAnode(); {the selection of the last responding node}
      send AA_Sel to selected AA node;
      start configurationTimer;

timeout(configurationTimer) →
      self-configure();
      CalculateNewHash()
      send S_Merge to all nodes;
      configured := true;
      skip;
```

Because each configured node in LHA has different independent functions, such as to allocate free addresses, detect the mergers or reuse the addresses of departing nodes, the functions of the "Configured" state are divided in this thesis into two orthogonal groups called "Assigning" and "Maintenance" states, which work concurrently. While the "Assigning" state represents the functions of direct assignment, allocation or search for free address in the network, the "Maintenance" state describes the functions that handle problems such as merging and partitioning networks. Following the basic idea of LHA, configured nodes are responsible for assigning addresses to new joining nodes. Because new nodes may send more than one request simultaneously, a configured node must be able to handle these messages at the same time. Therefore, the "Assigning" state of a configured node consists of two sequential states working concurrently, as shown in Figure 3-11 (parts I & II). While the states in part I are defined to serve new nodes by providing available addresses (if there are any) or searching for free ones (if there are none), the states in part II are required to modify the status of the assigning table according the receipt of New_Node messages (e.g. if a new node selects an available address provided from this table, the assigning status must be set, "assigned", upon receipt of New_Node from this node). In the figure the algorithm of configured node activates first the states, "Ready to assign" and "Idle of allocation", from parts I and II respectively. The detailed behavior of the configured node in the "Assigning" state is shown below in Algorithm 3.2 and the ensuing description of it after that.

Algorithm 3.2: Assigning algorithm of a configured node

```
const M;          {maximum number of free addresses in a node}
bool AA;          {init. false, the node is not used as an Address Agent}
bool Alone;        {init. Alone = false, a node is not alone standing}
int IP_status;    {values: 0->"free", 1->"pending" and 2->"assigned"}
int Av;           {current available free addresses in the node, init. AV=M}
int hop-count;    {increased by 1 for each hop of AA_A_Req, init. is 0}
int ID;           {the ID of the requester (i.e., Un-configured node) }
bool WA(ID)       {init. false, no need for an acknowledgement from this ID}
bool partition    {init. false, no need to activate the partition Algo.}

receive AA_Sol from Un-configured node∧ ~AA ∧ ~Alone  →
       if Av > 0 then
              SelectAddressAssignTable();
              IP_status :=1;
              Av := Av-1;
       fi
       start sendTimer(ID);{reserved for this requester (ID)}
       AddEntryToSendingList(); {add this timer to the list}

receive AA_Rep from configured node →
       if requester ID in sending list then
              if positive reply then
                     Stop sendTimer(ID); {the timer of this ID in sending list}
                     if requester ID in assigning table then
                            IP_status := 0;
                            Av := Av + 1;
                     fi
                     DeleteEntryFromSendingList();
              else
                     if requester ID not in assigning table then
                            Stop sendTimer(ID);          {in FSM -> TM_ws(ID)}
                            DeleteEntryFromSendingList();
                     fi
              fi
       fi

receive New_Node form a neighbor {not received before} →
       if new node IP is a successor then
              Stop alloTimer(ID);
              IP_status := 2; {assigned}
              WA(ID):=false;
       else
              if new node ID in sending list then
                     Stop sendTimer(ID);
                     DeleteEntryFromSendingList();
              fi
              if new node entry in assigning table then
                     IP_status := 0; {free}
                     Av := Av + 1;
              fi
              if sender ID in reverse list then
                     InactiveReversePathEntry();
              fi
       fi
       AddToConfiguredTable();
```

```
receive AA_Sel from Un-configured node ∧ ~AA →
       if sending list is not empty then
              Stop all sendTimer;
              DeleteEntryFromSendingList();
       fi
       AA := true;
       start searchTimer;
       send AA_A_Req to neighbors;
       newEntryReversePath();

receive AA_A_Req from Configured node ∧ ~AA ∧ ~Alone →
       if not received before then
              if Av > 0 then
                     SelectAddressAssignTable ();
                     IP_status :=1;
                     Av := Av - 1;
                     start sendTimer (ID);
                     AddEntryToSendingList();
                     AddEntryReversePath();
              else
                     if hop-count < hops threshold then
                            start sendTimer(ID);
                            hop-count := hop-count + 1;
                            AddEntryToSendingList();
                            AddEntryReversePath();
                     fi
              fi
       fi

receive AA_A_Rep from Configured node →
       if requester ID in reverse list then
              if AA=true {the receiver is working as AA for the new node} then
                     Stop searchTimer;
                     InactiveReversePathEntry ();
                     send AA_Conf to the new node;
                     AA := false;
              else
                     if requester ID in sending list then
                            if new node in assigning table then
                                   Stop sendTimer(ID);
                                   IP_status := 0; {free}
                                   Av := Av + 1;
                            else
                                   Stop sendTimer(ID);
                            fi
                            DeleteEntryFromSendingList();
                            InactiveReversePathEntry ();
                     else
                            if intermediate node then
                                   send AA_A_Rep to AA node;{based on reverse
                                   path}
                            fi
                            InactiveEntryReversePath();
                     fi
              fi
       else
              AddEntryReversePath();
```

```
                InactiveEntryReversePath();
        fi

timeout(sendTimer(ID)) →            { in FSM -> TM_ws  }
        if new node ID in assigning table then
                if the request is sent by an AA node then
                        send AA_A_Rep to AA node; {reverse path}
                        InactiveEntryReversePath();
                else
                        send AA_Rep to the new node; {positive}
                fi
                start alloTimer(ID);
                WA(ID):=true;
        else
                if requester ID in reverse list then
                        send AA_A_Req to my neighbors; {forwarding the message}
                else
                        send AA_Rep to the new node; {negative}
        fi
        DeleteEntryFromSendingList();

timeout(alloTimer (ID)) →                    { in FSM -> TM_allo  }
        FindEntryAssignTable();
        IP_status := 0; {provided address is available again}
        Av:= Av+1;
        WA(ID):=false;

timeout(searchTimer) →                { in FSM -> TM_search }
        if (LN > 0) or (there are some nodes with free addresses)
                partition := true; {this will trigger the "partitioning" state}
        else
                send AA_Conf to the new node; {negative}
        fi
        AA := false;
```

As mentioned above, the states in the "Assigning" state of a configured node are divided into two parts I and II, and the behavioral description of each of these parts as follows:

Part I:

Basically, the states of this part in each configured node are responsible for providing two kinds of services for new nodes (requesters) during the assignment process; first, the direct assignment of free addresses to the requesters (if the node owns free addresses) or, second, the search for free addresses in the network for the requester (if the node does not own any free address). However, if a node is standing alone (*Alone* = true) it must not provide any of these services because this is a special scenario of partitioning network scenarios, which will be described in Section 3.7.4 later. Algorithm 3.2 shows the LHA assigning algorithm of a configured node, wherein, the Address Agent (*AA*) variable refers to the search service of this node and its value will be (*AA* = true) when a node is in the "Search for free address" state; otherwise false. In addition, the *Alone* variable refers to the stand alone node when it is value (*Alone* = true); otherwise false. Basically, when a node is configured with a free address the new node enters the "Ready to assign" state, Figure 3-11, and the values of *AA* and *Alone* in this state are false. After that, it provides its services as follows:

- During the "Ready to assign" state, each configured node is able to reply with equivalent reply messages (AA_Rep or AA_A_Rep messages) on receipt of any address request message (AA_Sol or AA_A_Req) or to transmit an AA_A_Rep message over its reverse path to a requester. However, to reduce the network load due to simultaneous sending of multiple replies, LHA utilizes the Tm_ws timeout as described in Section 3.5.2. Thus, when a configured node receives a request message it selects a suitable timeout for sending a corresponding reply and then enters a "Wait to send" state. The detailed process for each event in the "Ready to assign" state are as follows:
    - When a node receives an AA_Sol message, see Algorithm 3.2, it checks if it is able to allocate a free address or not. The node calls the `SelectAddressAssignTable()` function in which if it has free available addresses (*Av>0*) it selects a free address from the assigning table. Notice, positive reply in Algorithm 3.2 refers to AA_Rep message if it includes a free allocated address. The node, then, selects a sending timeout and starts a timer (*sendTimer*) after which the node has to send the AA_Rep message. However, to ensure high likelihood of receiving the positive replies prior to the negative ones, the sending timeout value of a node having free addresses must be smaller than other nodes which have no free addresses, as presented in equation 3.10. After that, the timeout is saved in the sending list by using the `AddEntryToSendingList()` function which adds to the sending list a new entry identified by the new node ID and assigns the selected timeout Tm-ws to this entry.
    - When the node receives AA_A_Req message for first time (is not received before). In Algorithm 3.2 and depending on the *Av* value:
        - If *Av>0*, the node calls the `SelectAddressAssignTable()` function to select a free address and modifies its parameters according to that (sets the Seq state to "pending" and decrease *Av* by 1). Then, it inserts a new entry to the sending list by means of `AddEntryToSendingList()` function. Moreover, to record the reverse path[1] to the sender, the node uses `AddEntryReversePath()` function which adds new entry identified by the new node ID to a reverse list. Thus, LHA unicast messages can be transmitted over multiple hops. After that it starts *sendTimer* and enters the "Wait to send" state.
        - If *Av=0* the node first checks the message hop (hop-count) to avoid flooding a highly scalable network with the forward messages. Thus, the count of hop-count must not exceed a predefined threshold. If it is within the set range, the node increases it by 1 and calls `AddEntryReversePath()` and `AddEntryToSendingList()` functions to add an entry to the re-

[1] If there is a reply message (AA_A_Rep) the intermediate nodes can transmit the message over this reverse path. This mechanism is mainly used in the route discovery function of routing protocols such as AODV protocol [65].

verse list and sending list respectively. After that, it starts *sendTimer* (i.e. sets the Tm-ws timeout) and enters the "Wait to send" state.

- When the node receives an AA_A_Rep message and it has an active entry[1] for the requester in the reverse path list the node has to send the message to next hop on the reverse path. After that it marks this path as inactive by calling `InactiveReversePathEntry()` function. In the other case, when it has no entry, the node adds a new entry to the reverse list and signs this entry as inactive[2] by calling `AddEntryReversePath()` and `InactiveReversePathEntry()` functions respectively. This step is done to prevent, later, the node from handling any AA_A_Req message from the same initiator with identical sequence number.
- If a node does not have free addresses and receives an AA_Sel message, the node supposes that the new node has not succeeded in getting a free address from its neighboring nodes. This means that it has to work as AA node, so it sets its (*AA*= true) in order to serve the new node to get a free address from a configured nodes located further than one hop from the new node. After that, it broadcasts AA_A_Req message to its neighbors. Then, it starts a timer called *searchTimer* and enters the "Search for free address" state in which it waits for a response.
- A node leaves its "Ready to assign" and enters the "Idle" state when it discovers a stand-alone case (*Alone=true*) which indicates that there are no neighboring nodes within the transmissions range of the node. Because the task to detect and handle the stand alone case in LHA is a part of the "Maintenance" state (see tion 3.7.4), the assigning task must be idle which means the node is not able to allocate or assign addresses to other nodes. Basically, LHA prevents the stand alone node from assigning addresses in such a case because it may lead later (if there is a merger) to possible address conflicts in the network. Thus, in Figure 3-11 the value of variable (*Alone*) is set by the "Maintenance" state which works concurrently with "Assigning" state.

- In the "Wait to send" state, the node overhears the medium for reply messages (see AA_Rep or AA_A_Rep messages in Algorithm 3.2). Such an overhearing mechanism is needed to prevent sending out the unnecessary replies. LHA here allows a configured node to discard its reply message prepared to send to a certain requester when the node during the sending timeout overhears from any of its neighboring nodes another reply message for the same requester. Thus, LHA reduces the number of messages sent during the assignment time. However, to avoid the discard of possible reply messages which include a free address (positive reply in Algorithm 3.2) if a negative reply is received at first, the configured node which owns free addresses does not discard its reply message

[1] An entry in reverse list indicates that the node has received an AA_A_Req message.
[2] If a node marks an entry of an initiator in the reverse list as inactive, the node will not handle the initiator message including a sequence number that is identical to the saved one.

upon receipt of a negative reply. In the "Wait to send" state the node handles the following events:

- If the node overhears from its neighbors a positive AA_Rep message sent from a neighbor to a new node and there is an entry of this new node in the sending list as described in Algorithm 3.2, it stops the timer (*sendTimer*) of this entry and if there is an entry for the new node (ID) in the assigning table it changes the status of the allocated IP to "free" and increases *Av* by 1. In the other case, if a negative reply is overheard and there is an entry of the new node in the sending list the node only stops the timer if there is no entry for the new node in the assigning table. In this description the action to stop the timer means that the node discards its prepared AA_Rep message. Therefore, the node uses the `DeleteEntryFromSendingList()` function which deletes the entry of the new node from the sending list. Finally, if this list is empty it leaves its wait state and enters "Ready to assign".
- If an AA_A_Rep message is received, the node has to check the information of its reverse list which saves the reverse path information of this message. In the case where there is no entry, the node adds a new entry to the reverse list and signs this entry as inactive[1] by calling `AddEntryReversePath()` and `InactiveReversePathEntry()` functions respectively. As mentioned above, this step is done to prevent, later, the node from handling any AA_A_Req message from the same initiator with identical sequence number. In the other case, where there is an entry in the reverse list with information (ID) identical to that in the received AA_A_Rep message, the node checks the following:
  - If there is an entry for the new node in the sending list (i.e. an entry for sending AA_A_Rep or AA_A_Req), the node knows that there is another responder in the network that is trying to serve the requester. This means that there is no need to send the prepared message with identical information. Thus, if the node has allocated a free address to the requester (ID) the node discards its prepared AA_A_Rep message, stops the timer (*sendTimer*) and releases the allocated address by changing the status of the allocated IP to "free" and increasing the *Av* by 1. Otherwise, if the node does not have free addresses it stops the *sendTimer* in order to avoid the forwarding of the prepared AA_A_Req message. After that, the node calls `InactiveReversePathEntry () and DeleteEntryFromSendingList()` functions to sign the entry of the new node in the reverse list as inactive and to delete the new node entry from the sending lists respectively. Finally, if the sending list is empty it leaves the wait state and enters "Ready to assign".

[1] If a node marks an entry of an initiator in the reverse list as inactive, the node will not handle the initiator message including a sequence number that is identical to the saved one.

- If there is no entry of the new node in the sending list which means here that a request message from the initiator has been sent before, the node has to check the destination address in the received message. If the two addresses are identical and the reverse path is active the node knows that it is the corresponding node (intermediate node of the reverse path) and responds by sending the AA_A_Rep message to the next hop over the reverse path to the initiator (the AA node of the new node).

- If the *sendTimer* of a certain entity (ID) is reached and there is no allocated free address for the new node (in Algorithm 3.2, there is no entry for the new node in assigning table), the node has to check the reverse list; if there is no entry in the reverse list, the node responds by sending a negative reply of AA_Rep message, otherwise it forwards for this entry AA_A_Req. However, if there is an entry in the assigning table the node sends a positive reply message according to the requester; i.e. if the requester is the new node it sends AA_Rep message, otherwise, it sends AA_A_Rep message to the requesting AA node and signs the entry of the new node in the reverse list as inactive. In LHA, a positive reply message form any node requires acknowledgment (receipt of New_Node), so that the node sets WA(ID)=true (i.e. the node must wait for an acknowledgement from this ID) and starts allocation Timer (*alloTimer*) after sending AA_Rep or AA_A_Rep messages. Finally the node has to delete the information from the sending list by using the `DeleteEntryFromSendingList()` function and, then, if this list is empty it releases its state and enters "Ready to assign".
- If the node receives a New_Node message and the node is not the predecessor of the new node which sends the messages, it means that the selected address of the new node is not one of the free addresses set of this node. In this case, if there is an entity of the new node in the sending list the node discards the message which it had prepared to send to this new node. As described in Algorithm 3.2 the node stops the *sendTimer* and modifies the sending lists. After that, the node leaves the "Wait to Send" state and enters "Ready to assign" if the all entries in the sending list are empty.
- If the node receives AA_Sel message from a new node that has requested a free address, the node has to stop all sending timers (*sendTimer*) and delete their entities from the sending list. This means that the node must work as AA node (*AA = true*) to the requester and will only be responsible for doing so. In this way, LHA prevents duplicated searches performed by an AA node for different requesters at the same time. After that, the node sends an AA_A_Req message to its neighbors, starts the searching timer (*searchTimer*) and adds a new entry to the reverse list by calling the function `AddEntryReversePath()`. Finally, it leaves the "Wait to Send" state and enters the "Searching for Free Address" state.

- In "Search for free address" state (i.e. the node is working as AA for a new node where *AA*= true), if the node gets an AA_A_Rep message as a response to its AA_A_Rep mes-

sage, it knows that the responding node has offered a free address to the new node. Therefore, it sends an AA_Conf message including the free address to the new node. Then, it stops the *searchTimer* of this requester and calls `InactiveReverseSePathEntry ()`. After that, it changes its state and enters the "Ready to Assign" state in which *AA*=false. However, if the node does not receive an AA_A_Rep message during the searching Timeout, the node supposes that there are no free addresses in the networks and it behaves as follows:

- If *searchTimer* is reached and there are missing nodes or some nodes in the configuring list with free addresses, the node activates the maintenance algorithm by setting *partition* to true (i.e. start the partitioning algorithm of "Partitioning" state to check if there is a partition or not see Section 3.7.2 for more details). In this way the node may offer to the new node one of the now missing addresses. Therefore, the node leaves the "Searching for free address" state and enters the "Ready to Assign" state.
- If *searchTimer* is reached and there are no nodes with free addresses, the node knows that it is not possible to assign free address to the new node (maybe all free addresses in network are in the use). Therefore the node sends an AA_Conf message without any free address (negative reply) and changes its state to the "Ready to assign" state.

- In "Idle" state, which represents the case when the node stands alone (*Alone=true*), the node is not allowed to assign a free address to any requester. This case is a special scenario of the partitioning to avoid a quick reuse of missing nodes by this node. In this state and after a while, if the node gets information from the "Maintenance" state (*Alone=false*) that there is a reconnection with other nodes from its network, it has to release its state and enter again into "Ready to Assign" state.

Part II

In Figure 3-11, "Idle of allocation" and "Wait for acknowledgment" are the second part of states from the "Assigning" state, which are working concurrently with the other states described above. Basically, the two states are defined to enable every configured node to reallocate its available free address provided (allocated) to a new node (requester) when the requester does not select this allocated address as its own address. In this way, LHA allows the neighbors of a new node to offer simultaneously their free addresses without causing IP address exhaustion[1] or address conflicts later because these addresses are not selected by the requesting new nodes. In addition LHA increases in this way the success rate of the assignment of free addresses to the new nodes at the first attempt. The algorithm of both states is as follows:

- In "Idle of allocation" state, when WA of an (ID) is set true, which means that a positive AA_Rep or AA_A_Rep messages has been sent to the requester with this ID, the node

[1] IP address exhaustion is the term used to describe when there will be no more unallocated IP addresses available.

has to start the allocation Timer (*alloTimer*) and add an entry with this (ID) to the allocation list of timers (i.e. increase sL by 1). After that, it leaves this state and enters "Wait for acknowledgment" state.

- In the "Wait for acknowledgment" state the node providing a free address to a requester waits for an acknowledgment from this requester (New_Node) until *alloTimer* is reached. Here there are two possible scenarios as follows:
    - First possibility: it receives a New_Node message from the requester prior to the expiration of *alloTimer* (in Algorithm 3.2 see "receive New_Node" where the sender is a successor). The node stops the *alloTimer* of this allocated address and changes the status of this address in the assigning table into "assigned" (IP-Status = 2). Then, if the node has no other timers (sL = 0) for an offered address, it leaves this state and enters the "Idle of allocation" state.
    - Second possibility: the node does not receive a New_Node message during *alloTimer*. It supposes that the requester of the offered address has selected a free address offered by another node. Therefore, it releases the allocated address by changing the status of the offered address in the assigning table into "free" (IP-status = 0). Then it enters "Idle of allocation" state if all timers of offered addresses have been reached. Obviously, in MANETs the node may not receive the acknowledgment, which in turn may result in address conflicts if the address is assigned to another node. In LHA this situation has been studied and solutions have been defined to make LHA algorithm robust against abnormal scenarios as presented in next section, "Handling Special Cases".

3.5.5 Handling Special Cases

As discussed in the section of the assignment algorithm, LHA uses appropriate timers and scheduling mechanism to mitigate the effects of packet drops due to simultaneous sending of AA_REP messages by all neighboring nodes. However, in MANETs loss of messages may occur in other cases due to abnormal situations, such as interference, sudden node failure, bit error or erasures. Because LHA uses timeouts and repeat mechanisms, the loss of AA_Req or AA_Rep messages does not have an impact on the assignment process. However, this could be a problem in lost New_Node messages sent to the predecessors. Therefore, to make the LHA protocol robust against such situations the following two special cases are taken into account:

- Dropping of New_Node by some nodes including the predecessor (update issue):

When a new node broadcasts its New_Node message, the other nodes in the network have to update their tables to take account of the information received in the message. In the case of message dropping, LHA has an additional mechanism to provide the other nodes in the network with the required information about the new node. In LHA every Beacon message, see Section 3.4.2.5, includes information about the last two configured nodes in the network. In other words, each node adds to its Beacon the HHiDs of the last two configured nodes known by this

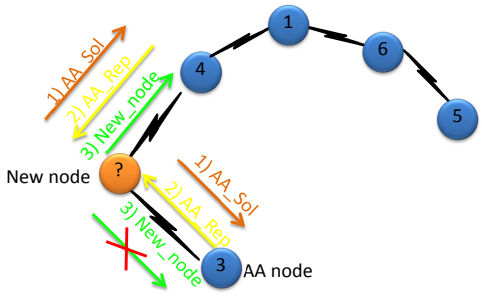
node and also the joining time of last node Jt_{last}. Thus, when another node receives a Beacon it compares this received information with its one saved in the configured nodes lists. From the comparison two cases can be defined as follows:

- If a received HHiD is one of its successors the node supposes that the New_Node message of its successor has been dropped. In this case the node adds the new node to the configured list and modifies the status of its IP successor in the assigning table into "assigned ", where IP-status = 2.
- In the second case, the received HHiDs are not equal to any of the HHiDs in the assigning table. This means that the new node is not a successor of this node. So the node has to check its configured list and it adds a new entry to the configured list if there is no entry of this address.

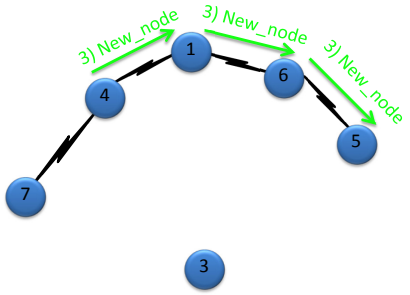
Although LHA is able to handle the update issue of broadcasting the New_Node message, it is preferable to ensure highly reliable packet delivery for LHA broadcast messages by utilizing the reliable broadcasting mechanism presented in [79], which is further work in which the author has been involved, but outside the scope of this thesis.

- Duplicate assignment issue:

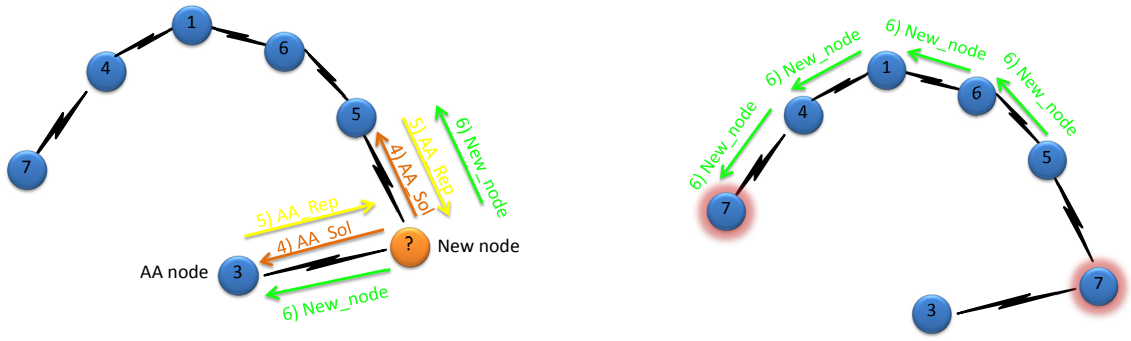
In MANETs there is a possible case when a predecessor does not get a New_Node message sent by its successor (the new requesting node) because it moves out of the transmission range as illustrated in Figure 3-12 (a and b). In this figure the assumption is that the new node selects the free address offered by node 3. However, node 3 does not get the New_Node message sent by the new node because node 3 moves away from other nodes in the network. As foreseen in the LHA algorithm, after a predefined acknowledgement timeout without receiving a New_Node message from the new node, AA node 3 offering a free address may use later (when the other available addresses are assigned) the offered address in the assignment process for another requesting node. This in turn may lead to duplicate assignments of the same IP address in the network as shown in Figure 3-12 (c and d).



(a) the provider (node 3) of a free address fails to get a New_Node message from the requester



(b) the predecessor (node3) of the new node7 moves away from the son and other nodes in the network



(c) node 3 provides the free address which is used by the first requester to another new node

(d) address conflicts due to both requesters using the address "7"

Figure 3-12: Duplicate assignment scenario

Let us suppose that the timer of node 3 expires and a new node enters the network from the right side as depicted in Figure 3-12 (c), where it can hear the nodes (3 and 5) which are not within the transmission range of each other. This means that one of the two neighbors (3 or 5) can assign a free address to the new node. Let us, also, suppose that the node 5 has no free addresses to assign and node 3 can only offer address "7" which has been offered before to another requester. This means that the new node will use the free address "7" offered by node 3. In this way the network will contain two nodes using identical addresses. LHA solves this problem in an efficient way, which is that the last joining node sends its New_Node message and this message will be forwarded to the node which has address identical to the one received in the message "7" as shown in Figure 3-12(d). In LHA this node (node 7) must release its address and start a new assignment process. In this way there will not be any address conflict because only one node in the network is using the address "7". Of course, when other configured nodes in the network receive the new node message they have to modify their configured lists to take account of the information of the last new node.

3.6 Network Merger Algorithms

The maintenance of configured addresses in a network is one of the essential tasks in address auto-configuration. In LHA, the "Maintenance" state in "Configured" state as shown in Figure 3-11 is designed to handle different auto-configuration issues, such as address conflicts during merging of networks or address management of departing nodes. As shown in Figure 3-13, this state consists of two sequential states called "Standby" and "Maintaining".

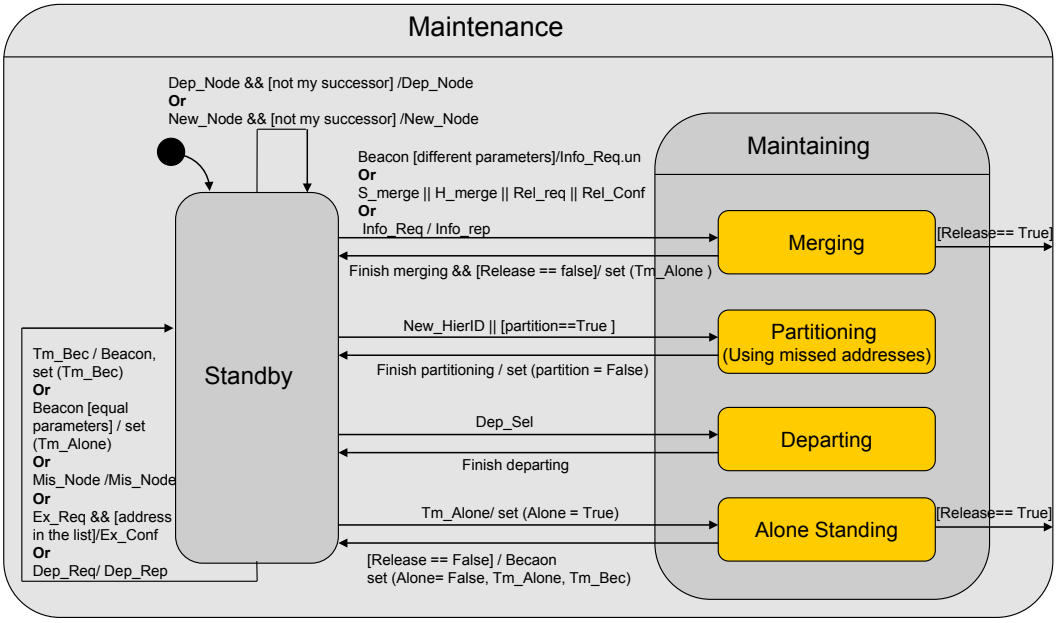


Figure 3-13: Maintenance state of LHA protocol

The "Maintaining" state in the figure includes four main states defined as "Merging", "Partitioning", "Departing" and "Alone standing" states. Initially, when a node becomes configured it enters the "Standby" state of the main state "Maintenance". In this state the node sends a Beacon message every Beacon Timeout (Tm-bec) and it updates the alone Timeout (Tm-alone) when it receives a Beacon from a neighboring node. The node in this state is responsible also for updating its database with recent LHA information carried in the Beacon, Dep_Node, Mis_Node or New_Node messages; for example the node adds new entry to the configuration list when it receives a New_Node message from a new node as described in Section 3.5.2. Moreover, in this state the node has to send an Ex_Conf message as a response for an Ex_Req message from a node detecting a partition and, also, it replies by sending Dep_Rep message if it receives a Dep_Req message from a departing node. However, if the node detects an event relating to any of the maintaining tasks as shown in the figure it releases its state and enters the corresponding state of the "Maintaining" state. In the following sections these events are described in detail. Basically, after handling the issue the node returns to "Standby" state as shown in the figure. In a certain case, after finishing the task, the node may not enter "Standby" state if there is a need for a reconfiguration process which indicates that a node has to release its current configuration to avoid any possible address conflict. To explain the working of the LHA algorithms in the "Maintaining" state here is first a presentation of the algorithm of "Merging" state, followed by sections on the algorithms of other states of the "Maintaining" state. However, to start with the basic idea of the merger, here are some useful terms.

3.6.1 Terminology

- "Border nodes": are configured nodes like other nodes in LHA but their positions are on the border of the network. Therefore, these nodes are assumed to be the first nodes which may receive a Beacon message sent by other border nodes of another network. Thus, the border nodes of a network are assumed to detect and handle the merger with other networks.
- "Unstable network": is a network which has a number of nodes less than or equal to a merging threshold (mer_Th). It is called unstable because its nodes have to release their configuration when their network merges with another bigger network (one with more nodes). For more details see Section 3.6.5.
- "Stable network": is a network which consists of a number of nodes more than mer_Th. In addition, the network which consists of more than one address hierarchy (resulting from the merger with other networks) is usually considered a Stable network.

3.6.2 Basic Idea

Because mergers may result in address conflicts in the network, conflict resolution is considered the main challenge here. Address auto-configuration protocols have tried to solve it to date by following different methods as presented in Section 2.4. LHA as presented in this thesis introduces a novel algorithm to handle the issue of merging networks. Contrary to centralized protocols, the basic idea of its merging algorithm is to allow every node in a network to be responsible for detecting any merger with another network and for handling possible address conflicts after the detection in robust actions with minimum effort. As mentioned in Section 3.4.2 each node sending a Beacon message has to include LHA parameters[1] referring to the configuration status of its network. Basically, if two networks merge together, the border nodes in the merger area of each merging network will be the first nodes which detect the merger, as shown in Figure 3-14 where nodes 6 and 17 are border nodes. In this figure, both networks are taken to be stable and there is at least one LHA parameter that differs between them. From the figure, when border node 17 receives from node 6 a Beacon message including parameters which differ from the saved ones, node 17 supposes that the sender is a border node belonging to another network. Thus, in Figure 3-13 node 17 releases the "Standby" state and enters the "Merging" state which includes four sub states ("Wait info reply", "Soft merging", "Hard merging" and "Reconfiguring") as shown in Figure 3-15.

[1] They are HierID, Hash, ROOT, the number of Merged Networks (MN), the number of configured nodes and the information $last_{conf}$ / Jt_{last} and S_last_{conf}) of the last two configured nodes in the networks.

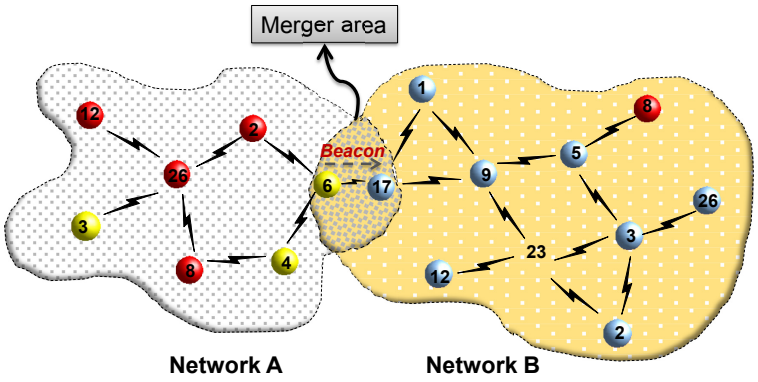


Figure 3-14: Merging of two networks

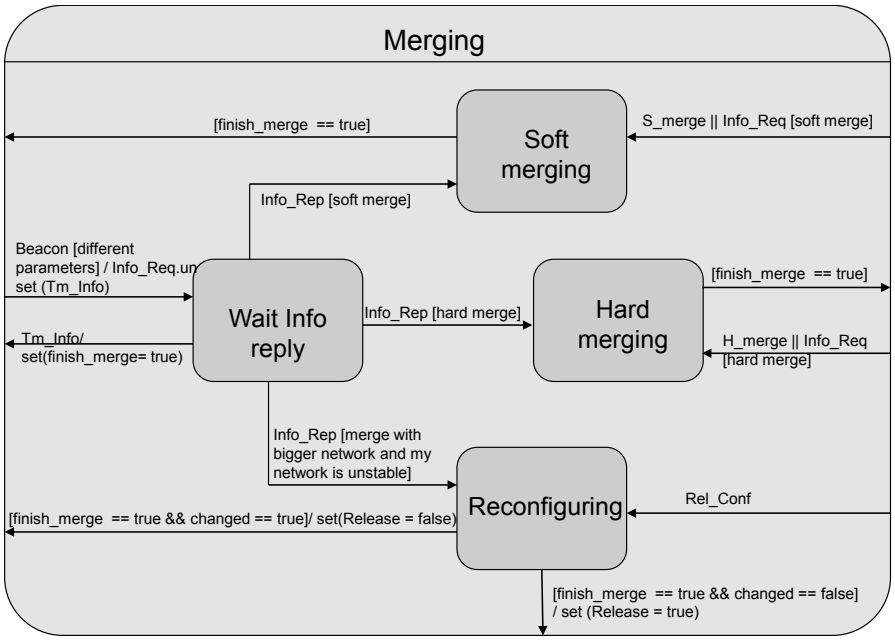


Figure 3-15: Merging state of LHA protocol

To show the process in the "Merging" state, a general scenario without special cases such as dropping packets is now presented as an MSC chart in Figure 3-16. Thus after the detection of a merger, the node sends an Info_Req message including the information and tables of its respective network and sets Info Request Timer (*Inf-Req-Timer*) with adequate duration defined as info Timeout (Tm-info). During this period, the node assumes that an Info_Rep message will be sent by the border node from the other network. In the case that the exchange of both info messages succeeds, both border nodes are able, after the comparison of the message information, to define the kind of the merger because both info messages include the network configuration parameters.

In the case of the merger of two stable networks, there are in LHA two main kinds of the mergers called Soft and Hard mergers. Depending on the HierID values of the merging networks the soft or hard case can be defined. Identical values lead to a Hard merger and different values lead to a Soft merger (the following sections describe both). The actions in each kind of the merger differ in the degree of modification to the configuration data in the nodes; however, each merger in LHA represents a uniform modification that every node from one network must follow to avoid any address conflicts after the merging. This means that if different border nodes from a network detect a merger with another network, they are able to define the possible address conflicts with the nodes from that network and, depending on the kind of the merger, select identical actions to solve them. So, after detecting and defining the kind of merger in Figure 3-16, both border nodes form the two merging networks enter the corresponding state (Hard or Soft) in the "Merging" state and then each border node is responsible for informing the other nodes from its network about the kind of the merger by sending a corresponding message, for example in case of Soft merger a Soft Merger (Soft_Merge) message must be sent in each network. Depending on the message received, the other nodes, in turn, enter the corresponding state in the "Merging" state and then uniformly do identical actions to complete the merger. In other words, all nodes from a network enter the predefined state from the border node and, thus, they make identical changes to its tables and data. This means LHA needs in normal scenarios only one broadcast message from the detecting node to handle the merging issues and this in comparison to other solutions is efficient and very fast.

However, to make LHA robust it was also seen as vital to discuss a special case when there is a merger with an unstable network. In this case it is not necessary to take either action (soft or hard merging). Here, there is only an action which must be done on the nodes from the unstable network, wherein, every node in this network has to release its configuration and start the reconfiguration algorithm which will be described later in Section 3.6.5. The reason is to prevent insufficient usage of a certain HierID by a network which consists, for example, of less than 5 nodes. In this way, the protocol avoids the unnecessary increment of the HierID numbers of the merging networks and, also, effort (signaling cost) is saved by avoiding configuration changes in a big network. In this case, the node in an unstable network enters a state called "Reconfiguring" state which is independent of the Soft and Hard merging states as shown in Figure 3-15. There follows a detailed description of the merging states, including the "Reconfiguring" state, together with a presentation of the powerful LHA solution to certain special cases, especially the case of simultaneous merging of more than two networks. This has long been considered a major challenge of merging scenarios and most protocols so far developed have failed to solve it, others have been weaker in their solution. Here, first, is the description of the soft merger.

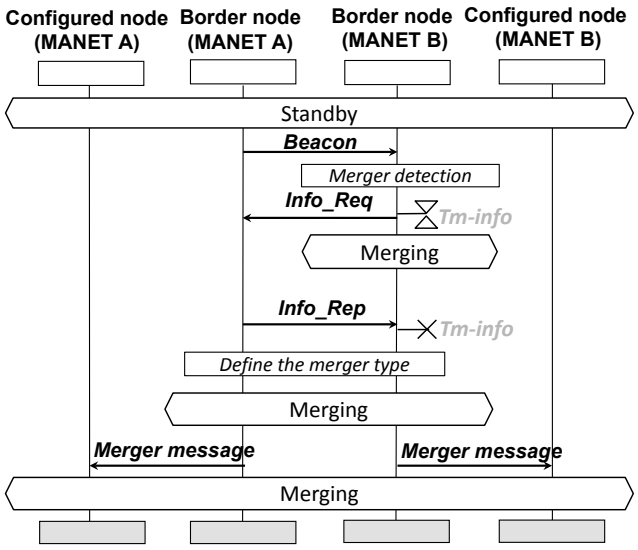


Figure 3-16: MSC chart of border nodes detecting the merging state

3.6.3 Soft Merger

As described in the basic idea of the merger, the soft merger presents the case where two or more stable networks having different HierIDs merge together. Because each HierID is a part of every IP address from a certain address hierarchy (see Section 3.2) the soft merger causes no address conflicts among the nodes after the merging. Although the soft merger does not cause address conflicts, the update process is required to handle later any merger between the new merged network which includes a variety of HierIDs. To understand the principle, here, in Figure 3-17, is an example of two simple stable networks (i.e. each stable network consists of only one address hierarchy). Each network selects its LHA parameters in which the HierIDs are different as depicted.

In accordance with the LHA algorithm, the detection of different beacons (including at least one LHA parameter differ from the saved one) leads both border nodes to exchange info messages including all tables and parameters as presented above in the merger basic idea. Because the HierIDs of both networks in the example are different, each border node discovers that there is a soft merger. This means that each border node has to enter the “Soft Merging” state from the “Merging” state as presented in Figure 3-15.

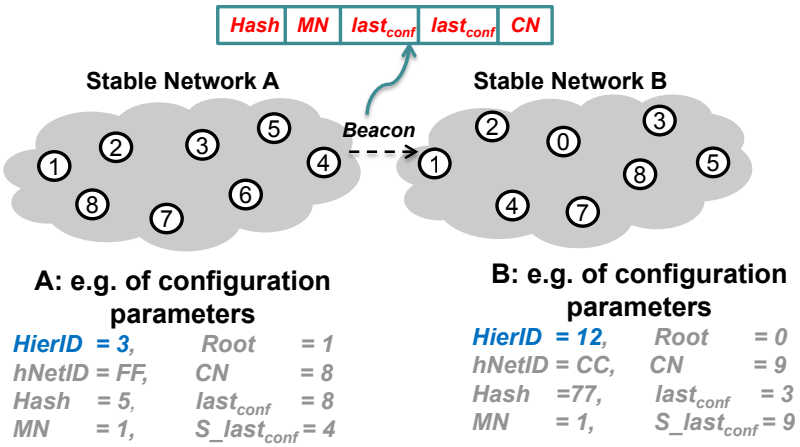


Figure 3-17: an example of soft merger (two stable networks with different HierIDs)

In LHA, each node in a "Soft merging" state is responsible for controlling the merger process in that the node has to update its parameters, save the information from the other network and inform the other nodes from its network about the merger. To permit this, the state consists of two sub states called "Update soft merging" and "Wait soft change" as shown in Figure 3-18. In the figure the "Soft merging" state is triggered by one of three events (Info_Rep, Info_Req or S_merge messages). The two info messages in this case (i.e. the messages sent by a node from another network with different HierID) are basically an exchange between two border nodes of different networks, while the S_merge message is sent inside a network by a sender from the same network of the receiver (i.e. both sender and receiver have identical LHA parameters). The procedure in this state can be described as follows:

- When a node is triggered by any of those messages it enters the "Update soft merging" sub state. The main action here is to add the information of the other networks to the merging table. Depending on the node information and the newly received information, the node updates its Hash and MN parameters. This will later help the node to detect the merging with other networks and define the kind of merger. However, to ensure a reliable merger process and avoid insufficient signaling to other nodes in the network due to the dropping of broadcasting messages, the node has to save a trace on its old Hash and MN numbers. After that, the node sets a merging timeout (Tm-mer) and broadcasts the S_merge message to its neighboring nodes. Finally, the node releases "Update soft merging" state and enters the "Wait soft change" state.
- In "Wait soft change", the node observes any Beacon sent by the neighboring nodes. During the Tm-mer timeout, if a node receives a Beacon message including information that is the same as the old saved information, the node supposes that the sender has not received the corresponding S_merge message. In this case the node has to resend S_merge message to this node. After the expiration of Tm-mer timeout the node knows that the soft merging state is finished, whereupon it sets the *finish_merge* parameter with true value. Thus, the node releases the "Merging" state from the

"Maintaining" state and enters the "Standby" state as shown in Figure 3-13. This means the node is able to act as usual by sending a Beacon message.

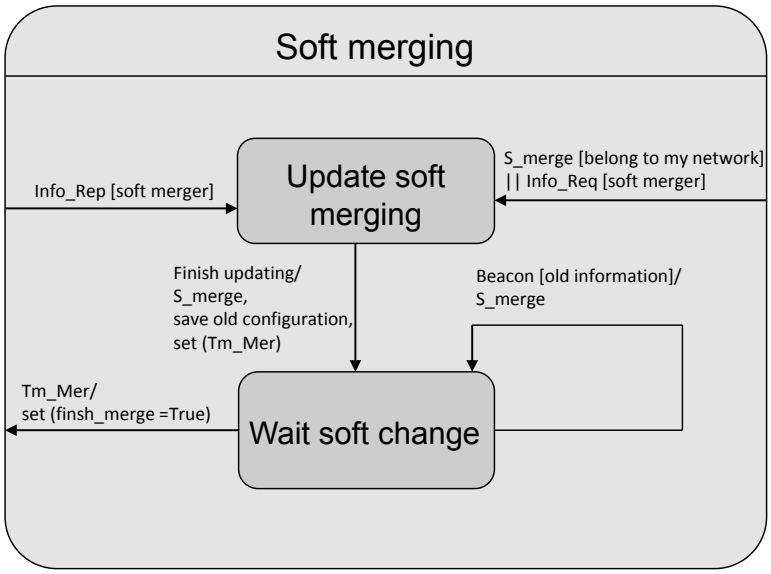


Figure 3-18: Soft merging state of LHA

3.6.4 Hard Merger

The algorithm of the "Hard merging" state is similar to that of "Soft merging", in that there are again two sub states as shown in Figure 3-19. However, the procedures of the sub states differ between hard and soft cases. This is because the hard merging deals with the merging of networks with identical HierIDs, which in turn leads to address conflicts among the nodes after the merger. Therefore, in LHA one of the networks (the one with the smaller number of nodes) has to change its HierID by choosing another HierID number which is free, i.e. not in use. Because this change is made to part of the IP addresses in the smaller network, the name given to this case is "hard". Basically, only the smaller network is in a "Hard merging" state, while the big one utilizes the new data selected by the smaller one to start the "Soft merging" state. As mentioned in the paragraph on the basic idea, each node in a network is able to detect and handle merging with another network. So, to avoid any conflicts in LHA when different nodes in the smaller network detect the merger simultaneously, every node in both networks selects the same new HierID by itself. In LHA the selection procedure used by every node is applied on both merger lists of merging networks. Because the merger list of any node in a network contains identical information belonging to the merging networks, the selection procedure based on this information enables every node to produce identical results after the selection (e.g. select the next available HierID in the list). This means that there is no problem if different nodes from a network detect and handle the merger case at the same time. The following describes how a node behaves in the "Hard merging" state:

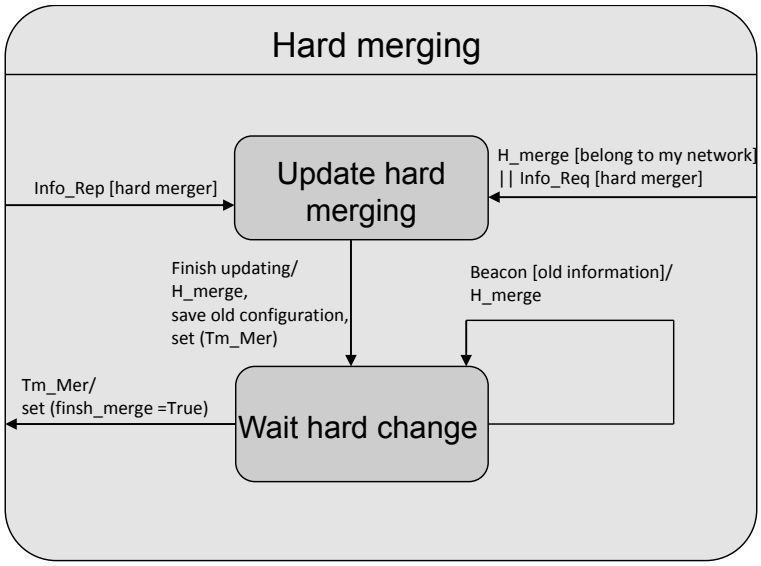


Figure 3-19: Hard merging state of LHA

From info messages (Info_Req or Info_Rep messages), when a node detects that it has to start "Hard merging" state because its HierID is equal to the one from other merging network and its network has smaller number[1] of nodes, the node enters the "Update hard merging" state as shown in the figure.

- In the "Update hard merging" state, a node knows that its network is small and it has to select a new HierID. Therefore, it builds a binary array of all HierID numbers, setting the array cells with value 1 and 0 for the occupied and available number respectively. Then, it uses a "simplest possible" strategy such as linear probing, or searching sequentially in the array until the node can find an available cell. Notice that it is possible for the search to wrap around from the last position to the first. Usually, the search starts upward from the HierID position of the node. After the definition of the new HierID number the node saves its old configuration and updates its table with the new data. Then it starts the merging timer with a value Tm-mer timeout. After that, it broadcasts H_merge message including all the information to all nodes in the smaller network. Finally, the node releases "Update hard merging" state and enters "Wait hard change" state.
- In "Wait hard change" (as in the soft merging state), the node has to ensure that the other nodes in the network have made the required change. Therefore, it observes the beacons sent by the neighboring nodes for Tm-mer timeout. During this period, if a node receives a Beacon message including information equal to the old saved info, the node supposes that the sender has not received its last H_merge message. In this case, the node has to resend the H_merge message to this node. After the expiration of Tm-

[1] If the network nodes are equal in number in both merging networks, the smaller hNetID must be used.

mer timeout the node sets the finish_merge parameter with true value and releases its state. The true value indicates that the node will also release the "Merging" state and the "Maintaining" state, and then enter "Standby" state.

3.6.5 Reconfiguration Algorithm

Because the algorithms of the soft or hard merger are achieved in each merging network by means of a broadcast mechanism, it is insufficient to use these algorithms in the case of merging with a very small network (Unstable network). The LHA algorithm deals with this case by using the reconfiguration algorithm, applied only on the smaller unstable network. The basic idea is that all nodes in the small network have to release their address configuration and get a new one from the big network. As there are few nodes in "Unstable network" the requesting nodes do not cause high signaling overhead. Moreover, the reconfiguration algorithm is designed to keep this cost to a minimum. Here follow the details of the algorithm.

As already explained above, the reconfiguration algorithm is selected when an unstable network merges with another network which has a bigger number of nodes (this may be an unstable or a stable network). Let us suppose that a stable network merges with unstable one as shown in Figure 3-20, where Y and X indicate stable and unstable networks respectively. In this case the nodes in the unstable network (X) have to release their configurations (addresses and network configuration) and then they have to get new configurations from the other merging network (Y) as shown in steps 1 and 2 in the figure.

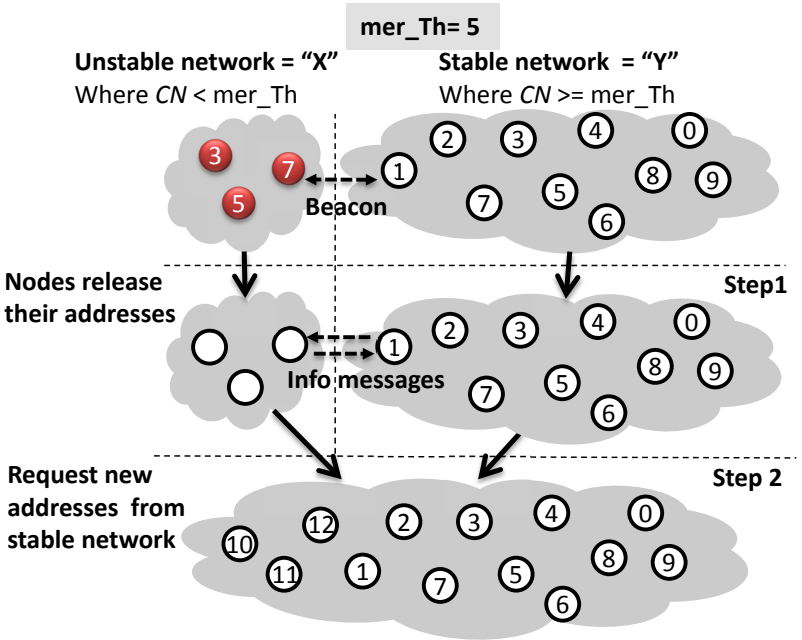


Figure 3-20: Merging scenario in case of two networks (stable merges with unstable network)

Depending on which border node first detects the merger, there are two scenarios:

- Say a border node of network X detects the merger first. This means that it receives a Beacon message sent by a border node from network Y. Therefore, this node follows the merging algorithm as follows:
    - It compares its parameters with the parameters in the received beacon. In the figure the node detects a merger with a stable network. Because of *CN*< mer_Th (merger threshold) the node knows that it has to release its configuration.
    - Then, the two border nodes exchange their information by means of info messages (Info_Req and Info_Rep). Therefore, the node that receives the Beacon sends Info_Req message including its parameters and tables to the other border node which has sent the Beacon (in this case, to the node from network Y) and waits for a reply.
    - Upon receipt of the request message the border node of network Y compares its data with the info received. The comparison indicates that there is a merging with an unstable network. The node, then, checks its assigning list. If the node has free address it sends Info_Rep message including its information and a free address which will be assigned to the border node of network X. Otherwise, it sends the Info_Rep message including only its information to the border node of network X.
    - When the border node of network X receives the Info_Rep message, it starts the reconfiguring state which consists of two states as illustrated in ure 3-21.
        - In "checking for change" state the node checks if the received Info_Rep message includes a free address or not as follows:
            - If there is a free address it saves the old configuration (e.g. the old HierID), assigns itself with the free address and changes its configuration data into the new received one. Then it sends Release Configuration (Rel_Conf) message to the other nodes in the network X and, also, sends New_Node message to the nodes in network Y. After that, the node set "true" value to a Boolean variable (*changed*) which indicates to the change in the configuration of this node. Then it sets a release change timeout (TmreCh) and enters the state "Wait for network change".

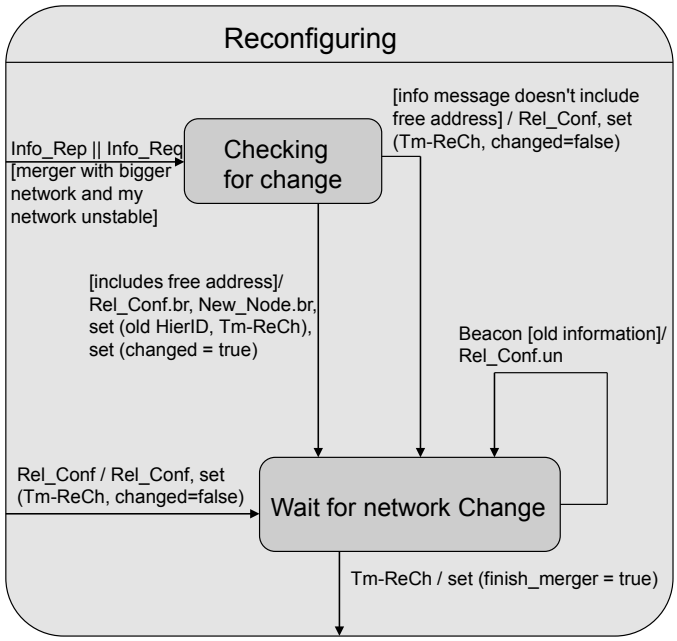


Figure 3-21: State machine of reconfiguring status

- If the Info_Rep message does not include a new address the node sets "false" to (*changed*) and sends a Rel_Conf message to the other nodes in the network X. After that, the node has to wait for a timeout "Tm-reCh" to ensure that the Rel_Conf message has been received by all nodes.

▪ In "Wait for network change" state, the node waits for the other nodes to complete the release of their old configuration. During the timeout (Tm-reCh) if the node receives from its neighbors any Beacon message including the old information it knows that this node has not received the Rel_Conf message. In this case it sends the message again to the sender of the old information. When the timeout period is passed, the node releases the reconfiguring status and checks the *changed* value. In the case of "true" value, the node resumes working as a part of the new network (network Y) with the new configuration. This means if the node receives an AA_Sol message it calculates the new address with regard to its new configuration. Otherwise, in the case of "false" value, the node begins a normal assigning algorithm like any new node joining the network. In this way the node gets a new address from any neighboring node of network Y. When the node receives the new configuration it can work as a part of the network Y.

o Upon receipt of Rel_Conf message each node from the network X has to release its configuration data and try to get a new one from any node from the network Y. However, before the node requests a new address from the new

network the node has to forward the message to other nodes in its network. To do that, the node enters "Wait for network change" state. As mentioned above, the node in this state waits for the duration to ensure that all nodes in its network have got the Rel_Conf message. Then the node release the "Reconfiguration" state and because *changed* =false in this case the node starts as a new node joining a network, as described in Section 3.5.

- The second scenario is that, the border node from the network Y detects the merger first. This means it detects that there is a merger with unstable network because of (*CN* < mer_Th). the algorithm ,then, will be as follows:
    - The node sends Info_Req message including its information to the border node of network X. In addition to the network configuration the message may include an allocated address if the node owns a free one.
    - When it receives this message, the border node of network X knows that there is a merger with a stable network and the nodes in its network have to release their configuration and get a new one from the new network. So it enters "checking for change" state and follows the algorithm as described above in the first scenario, the case where the node of X receives Info_Rep.
    - When the other nodes in the network X receive Rel_Conf message they will follow the steps as mentioned above in the first scenario (the steps of "Wait for network change" state).

3.6.6 Handling Special Case (Simultaneous Merging)

Most protocols so far developed that focus on the problem of merging networks study the case where only two networks are merging. However the merger of more than two networks at the same time is considered by these protocols to be a major challenge. The LHA algorithm introduces an efficient method of resolving these situations. To explain the LHA algorithm in the case of simultaneous merging of more than two networks, let us suppose that there are three networks, as shown in step 1 in Figure 3-22 where each cloud represents a network. In the first step it is assumed that none of the networks has merged before with another one. This means that the Hash numbers are selected randomly by the roots. Suppose r1, r2 and r3 are for MANET 1, 2 and 3 respectively, and the MN is set to 1 in every network; there is only one entry in the merging table.

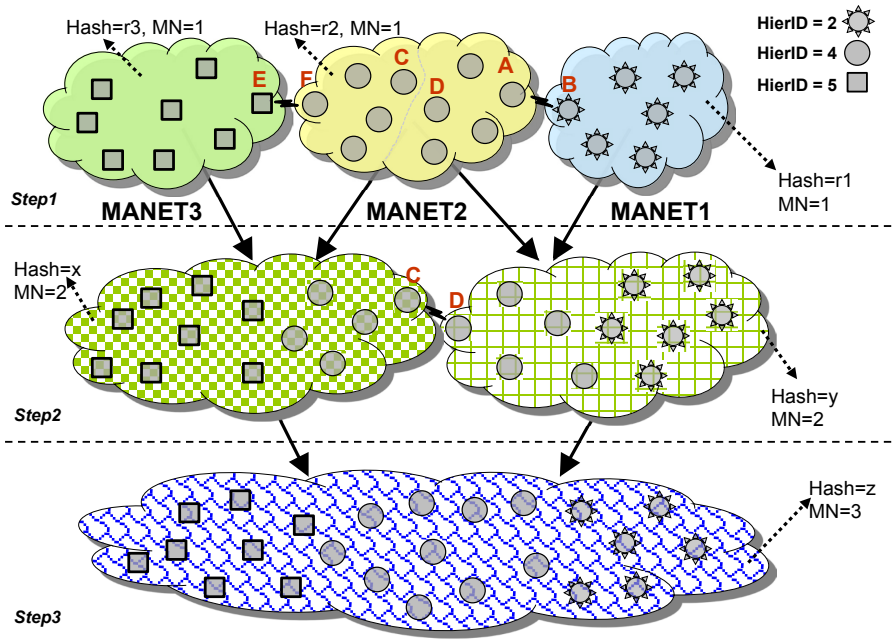


Figure 3-22: Simultaneous merging of three networks

In the figure each type of color-and-hatching in each cloud represents a distinct value of a Hash parameter used in this cloud. Also, each shape of a node represents a HierID number used by this node. Now let us suppose that MANET 1 and 3 merge with MANET2 simultaneously. The LHA algorithm of this case is as follows:

- When a border node, say A from MANET2 as in the figure, receives a Beacon message sent by Node B from MANET1, it detects a merger with MANET1 because LHA parameters in the Beacon from B are different. After that, the two nodes involved in the detection process exchange the information and tables of their respective networks, as presented in the basic idea of the merger.
- Upon completion of the message exchanges, each node calculates the new Hash and broadcasts a Soft Merger message (S_Merge) including the merging information to the other nodes. This message includes the information pertaining to the two merging networks.
- Basically in LHA, upon receipt of this message, the other nodes belonging to the network of the sender save this information in their table and calculate the new Hash.
- In the example, let us suppose that nodes E and F from MANTE3 and 2 respectively have detected a merger at the same time as with A and B nodes from MANET2 and 1. This results in a case that two different S_Merge messages from two nodes in MANET2 are sent at the same time. It is logical that some nodes of MANET2 will receive the merger message from A of the first merging (MANET1 with 2) faster than the message sent by node F of the second one (MANET2 with 3). We suppose that the

first receipt of the merger message in node D is coming from A and the first receipt in C is from nodes F.

- This results in the building of two networks (clouds) having different Hash parameters where node C with some nodes are in one cloud and other nodes with node D are in another cloud, as shown in step2 in the figure. Because C and D are on the border they will detect different parameters in the Beacon messages. In this case the border nodes (nodes C and D) of these two clouds will follow the same concept of detecting a new merger (the first, second and third cases of the merging algorithm as explained above).
- Finally (step3 in the figure), each node from the three networks will have only one Hash and MN number, which will later enable any node in the big cloud to detect mergers with other networks.

3.7 Network Partitioning Algorithms

An essential issue that may face address auto-configuration protocols in MANETs is that of network partitioning, in which one or a group of nodes may be unable to connect to other nodes in the network. The reason may be temporary obstacles among nodes or movement out of the transmission range of other nodes in the relevant period. The disruption of such connections among nodes may result in inaccurate management of address space because the information of some nodes is not available or not up-to-date during this period, which in turn may lead to address duplication if partitions reconnect with each other, especially in protocols with global assignment decision approaches as presented in Section 2.5.

Figure 3-23 shows this case where a network is divided into two partitions by the existence of an obstacle. In the figure let us suppose that after a certain time the missing nodes set {0, 1, 2, 8 and 9} is detected in partition 1 and missing node set {3, 5 and 7} is again detected in partition 2. Thus, if the addresses of the missing nodes in a partition are assigned to new nodes, there will be no problem in each separate partition. However, if the two partitions reconnect later there will be address conflicts as illustrated in the figure. So, during the design process the auto-configuration protocol must preclude later address conflicts from the assignment of missing nodes if the partitions reconnect. When this case is analyzed, two main questions can be defined as follows:

- First, when must a node detect a partition?
- Second, what is a safe method of reusing the address of missing nodes?

LHA is designed to handle the partition cases efficiently. The following sections describe the LHA algorithm in partition scenarios.

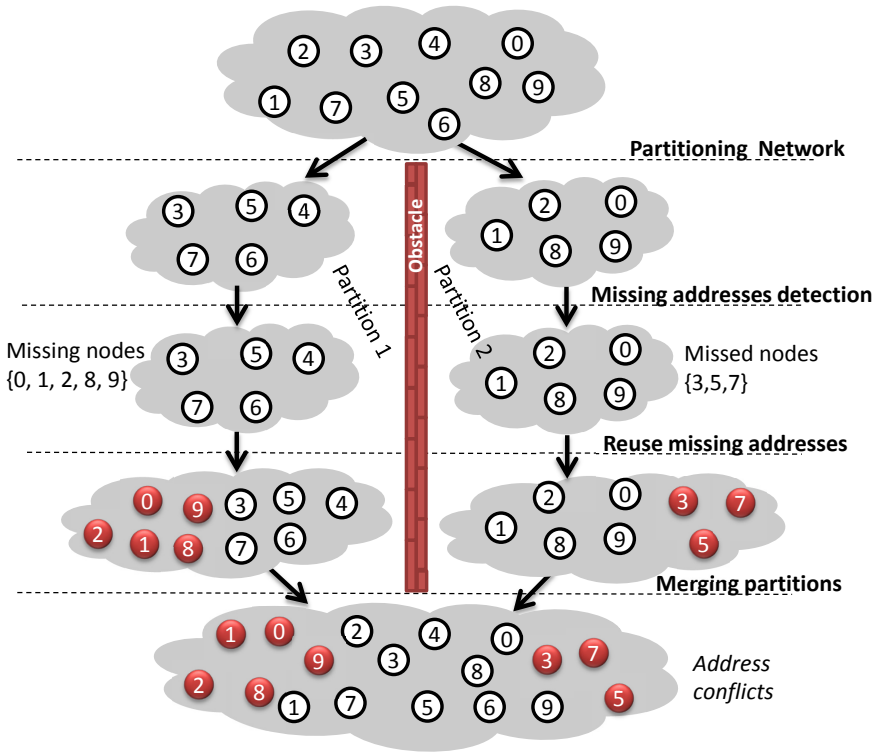


Figure 3-23: Partition network problem

3.7.1 Partition Threshold

Because LHA belongs to *local first decision* class there is no problem if two partitions of a network assign their free addresses and then they rejoin later, see Section 2.5. However, if each partition reassigns the addresses of missing nodes this may leads in address conflicts later. So, to fulfil the overall purpose, LHA partition algorithm will not be activated until there is a need to reuse those addresses. This method has a big advantage especially in situations of frequent partitioning and merging networks where the disconnection time is short.

In LHA, the partition algorithm will be triggered when an AA node in its "Searching for free address" state, as described in the joining algorithm, fails to find a free address in the network. In this case, there is a need to reuse the addresses of missing nodes when there is information about these addresses. Basically, the LHA partition algorithm is responsible for defining how those addresses are to be reused without causing address conflicts if later the nodes of different partitions reconnect. The LHA method of solving this case is to change the configuration parameters of every node existing in the partition. This means that when a partition decision is made, the nodes are instructed to change their configuration data. To prevent frequent configuration change (which would require additional communication cost among connecting nodes), LHA defines a partition threshold after which the network starts handling the partition problem. The way this threshold is selected is that LHA does not make a change in a partition if the number of Lost Ad-

dresses[1] (*LoA*) compared to the current number of configured nodes (*CN*) existing in a network is very small. This is logical because in a big partition the advantage of reusing some addresses, e.g. five lost addresses, will be very low compared to the cost of changing the address configuration of many nodes, e.g. more than 500 nodes. From this, the partitioning algorithm starts when the comparison in equation (3.11) is true, where $HHID_b$ is the number of bits of the HHID block.

$$LoA + CN \geq (2^{HHiD_b})/\mathrm{M} \quad \text{and} \quad \frac{CN}{LoA} \leq 2 \tag{3.11}$$

In LHA the *LoA* number is obtained from an equation (see equation 3.13) based on the sum of all weights of the Lost Nodes (*LN*) in the network. Let ln_i be the lost node *i,* basically, each lost node is assigned with a weight W where $W(ln_i)$ will indicate to the weight of the lost node *i*. The weight of node i $W(ln_i)$ depends on the number of its not allocated addresses ($n^i_{not\_alo}$) and a hierarchy level deference (HL_d) calculated from, $HL_d = HL_{max} - HL_{ln_i}$, where:

- Hl_{max}: is the maximum number of the levels in an address hierarchy, where $M^{Hl_{max}} \leq 2^{HHID_b}$.
- Hl_{ln_i} : is the hierarchy level of a lost node i (ln_i) in a network; where $0 < i < LN$ and $Hl_{ln_i} < Hl_{max}$.

In other words and in more detail, a node as defined in LHA owns a block of available addresses and as soon as a node is lost, its free addresses (which are not allocated to other nodes yet) will be considered lost addresses. Because LHA uses a hierarchal address structure, the possible number of lost addresses will be equal to or bigger than *LN*. So, to calculate *LoA* from the actual lost nodes, the node, which starts the partitioning algorithm, has to define the weights (*W*) of every lost node as shown in equation (3.12). This equation represents the weight value $W(ln_i)$ of a lost node i (ln_i).

$$W(ln_i) = 1 + n^i_{not\_alo} * \sum_{l=1}^{HL_d-1} (\mathrm{M}^l) \tag{3.12}$$

Finally, the sum of the weights of all lost nodes will refer to *LoA* as given in equation (3.13).

$$LoA = \sum_{i=1}^{LN} W(ln_i) \tag{3.13}$$

[1] The number of lost addresses is related directly to the lost nodes and indicates the sum number of all not allocated addresses (n_{not_alo}) in the hierarchy which can be managed by those lost nodes and their successors.

3.7.2 Partition Algorithm

In LHA the "Partitioning" state of a node may be triggered by other nodes or by the node itself. In Figure 3-24 a node may receive a New_HierID message from a neighboring node or if a node in "Searching for free address" state is unable to find a free address, as mentioned in Section 3.5.4, it sets (partition = true) which triggers the "Partitioning" state. Responding to either of these two events, the node enters the "Checking" state in which it has to check the partition threshold and update its table according to the comparison decision. Possible cases in the partition algorithm are as follows:

- In "Checking" state, two actions are possible based on the comparison of the threshold in "Equation (3.11)".
    - If *LoA* >= partition threshold, LHA supposes that there is a partitioning in the network which means that the node is able to use the addresses of the missing nodes in its partition. So it sends an AA_Conf message (positive reply) including an address of a missing address with the new HierID. Then the node enters "Updating" state.
    - If *LoA* < partition threshold, the node knows that there is no possibility of assigning free addresses to the new requesting node. The node, then, sends an AA_Conf message with no free address (negative reply) to the new node. After that, if there are no nodes with free addresses the node releases the "Partitioning" state and enters "Standby" state; otherwise, the node sends an Existence Request (Ex_Req) message including a list of the nodes which have not assigned their free addresses. Then, it sets an existing node timer Tm-exist and enters the "Recovery" state which represents an address recovery algorithm, see next section for more details.
- In "Updating" state the node has to update its tables by changing the status of all missing node into "available" and, then, it sends New Hierarchal ID (New_HierID) message to inform other nodes in the network about the new HierID by which the using of lost addresses will be safe. Then it sets the change partition timer Tm-par and enters "Wait partition change" state as shown in Figure 3-24.
- During "Wait partition change" state, the node analyses any Beacon message received from a neighboring node and defines if there is a need to resend a New_HierID message to the sender because it fails to change its HierID to the new one. In this case LHA avoids unreliable using of broadcasting messages and ensures that each node has made the required change sent in New_HierID. After the change timeout the node releases the "Partitioning" state and enters "Standby" state, which means that the partition algorithm has been finished.
- In "Recovery" state, LHA updates its tables on each receipt of an existence confirmation (Ex_Conf) message. After the expiration of Tm-exist and before the "Partitioning" state is released, the nodes broadcast a Mis_Node message if any corresponding

node in Ex_Req message does not respond with the Ex_Conf message. The next section describes this case (the address recovery algorithm) in detail.

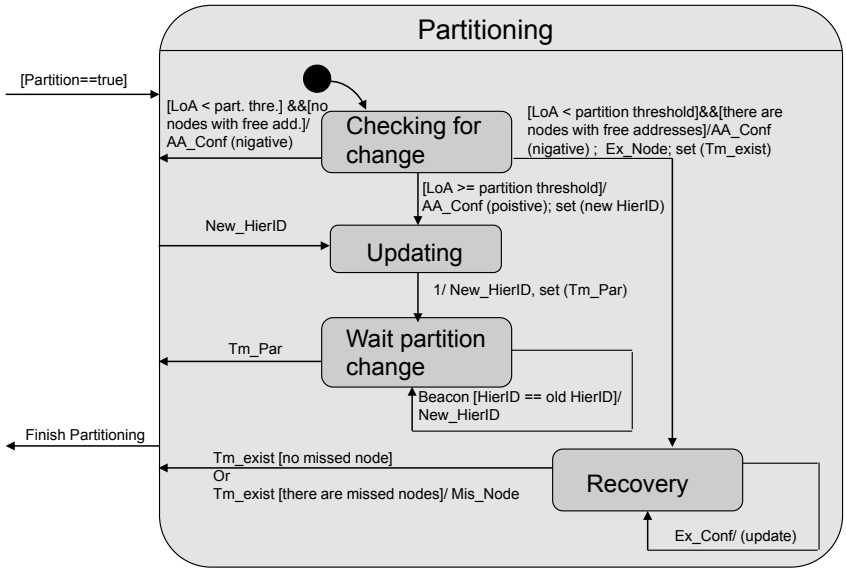


Figure 3-24: Partitioning state of LHA

3.7.3 Address Recovery Algorithm

As mentioned above, the "Partitioning" state will not be terminated if the threshold of the lost addresses is not reached and, at the same time, there are free addresses with some nodes which still exist in the configured list. LHA in this case supposes that there may be some missing nodes in the network but there is no information about them due to the failure to detect of those nodes. Therefore, the "Partitioning State" starts the address recovery algorithms to seek the existing nodes of which some or all free addresses are not allocated to other nodes. From the responses from those nodes, the partition algorithm gets accurate information about the missing nodes in the network. Because only the nodes which are expected to have free addresses are requested, LHA utilizes a multicast message which in turn reduces the signaling cost in the recovery process. The MSC chart of this case is presented in Figure 3-25. From the figure the algorithm can be described as follow:

- The node may, in the "Partitioning" state, detect in its configured list the existing of nodes which may own free addresses. This process in LHA is possible because LHA function divides the addresses space among nodes which in turn enables each node to define the successors of other nodes in the network. After that, the node sends a multicast message called the Existing Request (Ex_Req) message to the selected nodes.
- Upon receipt of this message by a corresponding node it responds by sending an Exist Confirmation (Ex_Conf) message to the requester.

- The receiving of an Ex_Conf message by the requester indicates that this node is still connected to the network and that it has not assigned its free addresses to other nodes because of packet drops or other issues such as a temporary obstacle.
- At the end of the existing timeout, if there are some nodes with no response, the requesting node assumes that these nodes are missing and there is a need to inform the other nodes in the network of that. Therefore, it sends a missing node message in the network. After that it updates the information of its tables and releases the "Partitioning" state and enters the "Standby" state.
- When other nodes in the network receive a Mis_Node message they update their tables according to the new information.

This mechanism to detect the missing nodes is not the only one in LHA because it starts only when there is a need to assign free addresses. In the departing algorithm, LHA uses another method to detect the missing nodes, for details see Section 3.8.

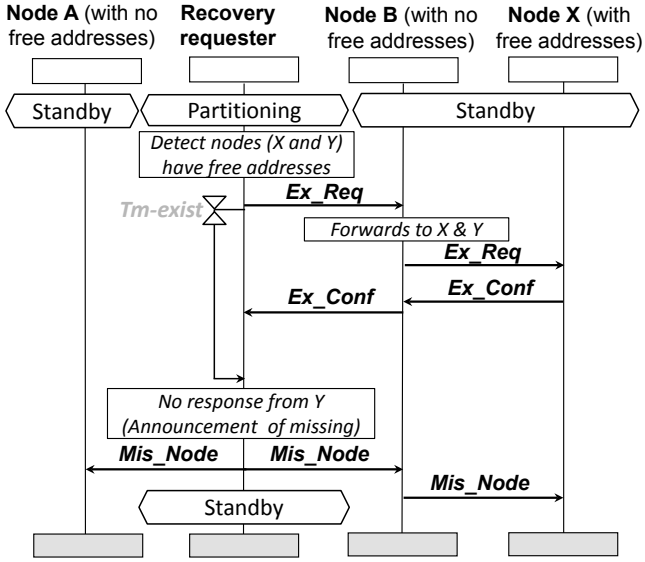


Figure 3-25: Search for nodes X and Y which own free addresses wherein node Y is no more in the network

3.7.4 Special Case (Stand Alone Node)

A special case of a partition scenario is that when a single node is not able to connect other nodes in the network, for example because of an obstacle. The solutions in much of the literature usually consider this case as an abrupt departure. The solving of this case as departing node may, however, cause address conflicts in the network because the node may reconnect to the network when the obstacle vanishes.

The LHA protocol defines an "Alone Standing" state in which this problem is well defined and solved as shown in Figure 3-26. The basic idea in LHA to solve this case is to change the

configuration of this node to a temporal or preserved one in which a node is not able to assign addresses form the original network configuration to a requesting nodes. There follows a detailed description of the state machine in the figure.

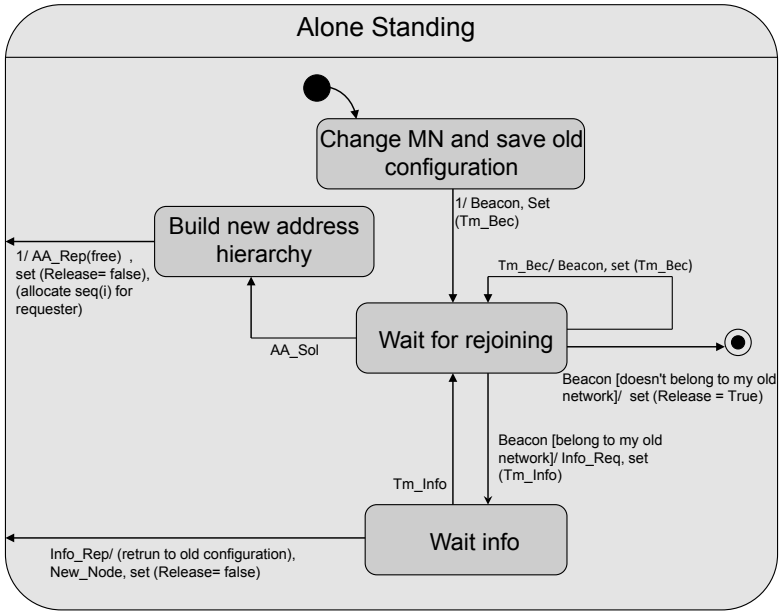


Figure 3-26: State machine of a node with alone standing state

The "Alone Standing" state is triggered when a node in "Standby" state (see Figure 3-13) detects an event that a node is standing alone. LHA generates this event when the standalone timer expires without receiving any Beacon message from neighboring nodes. This means that during the Tm-alone timeout the node has failed to detect any Beacon from its neighbors. So in this case the node concludes that it stands alone (*Alone=true*). Thus, in the "Alone Standing" state the node behaves as follows:

- In "Change MN and save old configuration" the node has to change its MN to a preserved one or 0, and to keep a track of its configuration (saving its original configuration). Then, it enters "Wait for rejoining" state.
- In "Wait for rejoining" state the node works in a normal way by sending regular Beacon message including the new MN. During this state the new node may receive one of two messages; Beacon from another configured node or AA_Sol message from a new node.
    - If the node receives a Beacon message from another node it has to do the following:
        - If the parameters in the received Beacon are equal to the old ones (HierID, Root, Hash and MN) then it reuses its old configuration. Then it sends an Info_Req message with empty configuration and its lists. After that, it enters "Wait Info" state.

        - If the parameters in the received Beacon are not equal to the old ones it has to release its configuration. In this case the node sets (Release=true) and leaves the "Alone Standing" state in order to start a new address assignment algorithm by sending an AA_Sol message as described in Section 3.5.
    - If the node receives an AA_Sol message from a new node it releases its state and enters "Build new address hierarchy" state.
- In "Build new address hierarchy" the node configures itself as a new root as described before. This means that it chooses a random HierID and HHiD. Then it defines the network parameter, as presented above, by using its MAC address. After that it sends an AA_Rep message including its configuration and the new address which will be assigned to the requester. Finally, it sets (Release=false) and then releases its stand-alone state and enters "Standby" state. It should be noted that when the requester receives this message it will not discover that this is a stand-alone node. So it will be a new member of this new address hierarchy and it sends a New_Node message to its predecessor (the root node in this case).
- In "Wait Info" and upon receipt of the Info_Rep, the node uses the information of this message to update its tables. After that it checks if its HHID has been noted as a "missing" one. If not "missing", it will continue as normal node but in the other case the node will be obliged to inform other nodes in the network about its reconnecting. Therefore, it sends a normal (New_Node) message to all nodes in the network. Upon receipt of this message, other nodes update their tables according to the new information. This means that if there is an entry of the sender in the configured nodes list (Table 3-2) the receiver checks the status of the sender in the table; so if the status is "available", the receiver will not do anything because the sender has a normal configuration and if the status is "missing", the receiver changes the status to "available". Otherwise, i.e. the information of the New_Node message is actually new, the node will deal with this message as described in section "joining new node" (update configuration table).

3.8 Node Departure Algorithms

A node in a MANET is able to join and release its network. When a node wants to release the network, the first issue is how other nodes in the network must handle the address of this node. Moreover, if the departing node has available addresses which are not allocated to other nodes the other issue here is which node in the network should be responsible after that for managing them.

LHA handles these issues by allowing a neighbor of the departing node to work as Departure Agent (DA) node which will in turn decide by itself a suitable way to deal with the free address-

es. It may manage them by itself or give the responsibility to another node in the network. The departing algorithm in this case is divided in two parts as follows.

3.8.1 Departing Node Algorithm

The LHA departing algorithm is triggered by a node when it needs to leave its network for such reasons as the need to join another network or to switch off for saving energy. The departing node, in this case, has to search among its neighbors for a DA node which may be able to manage the free addresses if the predecessor of the departing node does not exist in the network. In LHA a predecessor has the priority of managing the addresses of its successors. Basically, the DA node must be the predecessor of the departing node if it is within the transmission range of the departing node, but, if not, any neighboring node is able to work as DA. The selection phase of the algorithm fulfilling this principle is described in Figure 3-27 as follows:

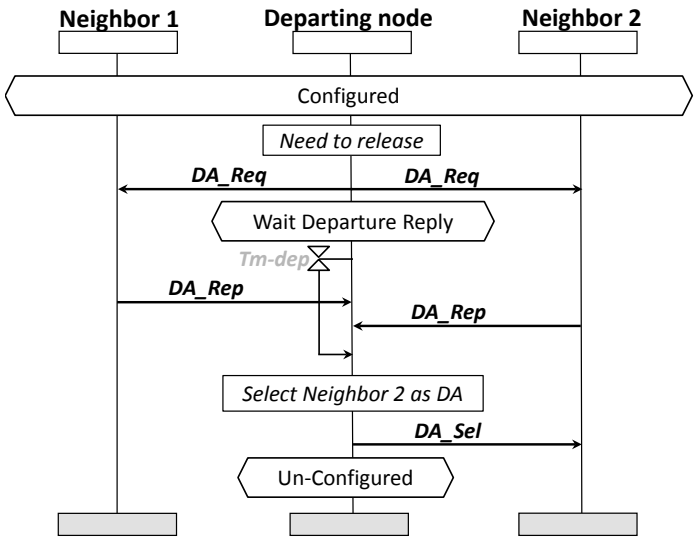


Figure 3-27: Selection of DA node by the departing node

- When a node wishes to depart from its network it sends a Departure Agent Request (DA_Req) message to its neighbors and starts departure timer Tm_dep. This message includes information about the predecessor and successors of this node.
- Upon receipt of this message, the neighboring nodes respond by sending Departure Agent Reply (DA_Rep) message to the node. This message informs the departing node about the readiness of the neighboring node to serve this node in the departing process.
- After collecting Dep_Rep messages, the departing node has to select one of its neighbors; as mentioned above, the priority lies with the predecessor if it is one of the neighbors. The figure shows a scenario where node 1&2 are not the predecessor of the departing node but node2 is selected because it has fewer free addresses than node1. After that, the node informs the DA node by sending a Departure Agent Select (DA_Sel) message, in-

cluding information about the selected node. Upon receipt of the DA_Sel message by node2 the selection phase is finished and then the node releases the network without any problem.

3.8.2 Departure Agent algorithm

Because the management task is given to DA node of the departing node, the algorithm describing this case is illustrated in Figure 3-28, presenting the state machine of a node receiving a DA_Sel message. The algorithm of DA starts as follows:

- When a node receives DA_Sel message the node enters "Checking and updating" state. In this state there are two cases:
    - If DA is the predecessor of the departing node:
        - It removes the address from its configuration table and it changes the status of this successor in the assigning table to "free". This means it can assign this address directly to any requester.
        - Then it broadcasts Dep_Node message in the network and releases the departing state.
    - If DA is not the predecessor of the departing node it has to discover the existence of the predecessor in the network because that node has the priority of managing the IP addresses of its successor. Therefore DA completes the following steps:
        - It sets Manage_status parameter to 1. This means that it is responsible for managing the IP address of the departing node during the time for searching for the predecessor.
        - It sends Departure management (Dep_Manage) message in the network and starts the management timer Tm-manage.
        - Finally it releases the "Checking and Updating" state and enters "Wait Management" state.
- In "Wait Management" state DA waits for a response from the predecessor of the departing node. Here, there are two cases:
    - If the timer expires, the node supposes that the predecessor is missing. Therefore the node sends Predecessor Existence (Pre_Ex) message in the network and starts the exist timer Tm-exist. After that it releases its state and enters "Wait Exist Ack." state.
    - If the DA receives a Dep_Node message from the predecessor of the departure node. This means that the predecessor exists in the network and it will take the responsibility of managing the IP address of it is successor which is the departure node.

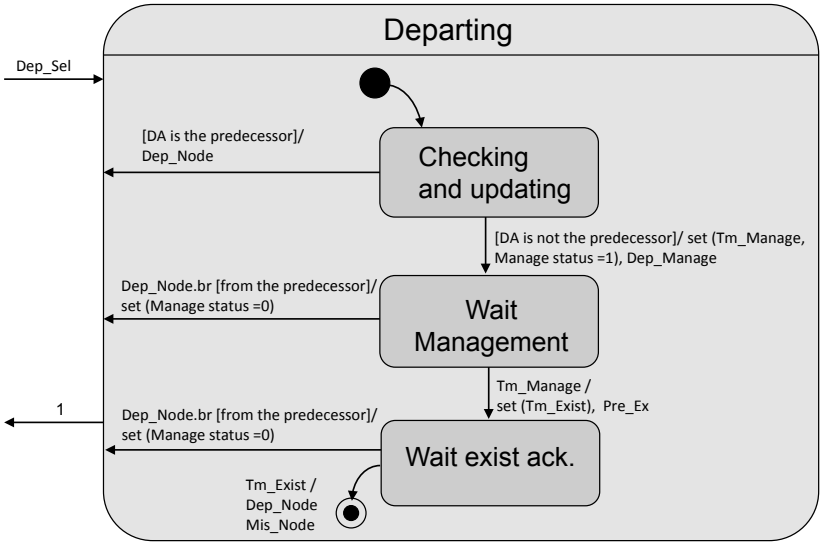


Figure 3-28: a state machine of a node working as Departure Agent (DA)

- In the "Wait Exist Ack." State, the DA node which is not the direct predecessor of the departing node assumes that the predecessor is not in the network but it needs to be sure before the announcement of this information. So it waits a response from the predecessor. Here, the possible cases are:
    - Dep_Node message sent by the predecessor means that the node is not able to manage the address of the departing node because the predecessor of the departure exists in the network. In this case the node has to set its Manage status to 0 and modifies its table according to new information. Finally, it releases "Departing" state and enters "Standby" state of the "Maintenance" state.
    - If the Tm-exist timer expires, the node ensures that the predecessor is missing and there a need to inform other nodes in the network of that. So it sends two messages, Mis_Node and Dep_Node, in the network to inform other nodes that the predecessor is missing and that it is thus responsible for the IP address of the departing node. After that it releases the "Departing" state and enters the "Standby" state.

Chapter 4 Performance Evaluation

This chapter is mainly an evaluation of LHA using a well-known simulation tool (ns2) [80] .Ns2 is the tool selected because it is the most popular one in academic use [81], especially for wireless ad hoc research, and has open source and a rich components library. Moreover, because of its well implemented IEEE physical/MAC layers, many ad hoc protocols have been developed and tested with the ns2 simulator.

This section provides a detailed and comprehensive performance evaluation of LHA compared to three counterparts, namely MANETconf, Prophet and Buddy. The first reason for selection of those protocols is that they are stateful protocols which, like LHA, utilize a distributed assignment process, as presented in Section 2.3. However, the other reasons behind the selection in each case are as follows:

- MANETconf is selected because it uses an assignment process with a global overview mechanism. This allows MANETConf during the assignment process to collect the status of other nodes in the network, which in turn, makes this protocol able to handle most addressing issues in MANETs, such as abrupt departing nodes and partitioning/merging networks.
- Unlike MANETconf, the Prophet protocol is not able to solve issues such as node departure or network partitioning because it has only a local overview. However, Prophet has been chosen because it operates the fastest address assignment process among all known stateful address auto-configuration protocols due to its simple handshake for address allocation based only on local assignment by a distributed method, as presented in [82] and [83]. Furthermore, the number of messages in Prophet is kept at a minimum compared to other protocols which have been developed. Because of these features the authors in [84] have skipped the comparison with Prophet, although they detected from their quantitative analysis that the performance was likely to be better than that obtained from their protocol with its cluster-based approach.
- The Buddy system, on other hand, represents the basis design of most stateful protocols using local assignment decisions with allocation tables; as is the case in [51], [52] [53] [54] and [55]. The protocol presented in [52] is selected as a representative of Buddy protocols because its assignment process requires less signaling cost than the protocols in [51] and [53], and it is not based on any routing protocols as are the protocols in [54] and [55] which depend on the Optimized Link State Routing (OLSR) [58] protocol. Moreover, it shows an even distribution of free addresses among nodes because of the rule that a new node should select an agent node which owns more free addresses. Furthermore, its search function for free addresses is economical because it uses hop-by-hop search. In

addition, it solves certain critical issues such as message drops, abrupt node departure and network partitioning/merging. Finally, the functions and algorithms of the address auto-configuration are well described in this protocol.

Basically, in auto-configuration protocols the unique address assignment function of joining nodes is the core process. A node is, namely, unable to communicate until it obtains the network configuration. Also, this joining process happens frequently and in several cases may lead in high signaling cost and latency if an inappropriate mechanism is used. Therefore it constitutes one of two main challenges of those protocols. The other big challenge of auto-configuration protocols is the function of the resolution of possible addresses conflicts resulting from networks merging. Most researchers are, thus, mainly concentrating on developing those two functions. It makes sense to measure the performance improvement resulting from both these functions when considering LHA. This means concentrating on the unique assignment function of joining nodes and on the address conflict resolution function when networks merge. Because LHA and Buddy are the only protocols using a multi-hop assigning function it is important to analyze this function of both algorithms before starting on the performance evaluation of the address assignment function of all protocol in different scenarios. So, first the behaviors of both protocols must be explained in a simple scenario which may assist for understanding their results in complex situations. Concerning the resolution function of merging networks, we will study that only for LHA because (in contrast to other protocols) LHA addresses the problem by sending only one broadcast message in the network. In this method, a node detecting a merger is able to handle all the conflicts in the network when it sends a single message. So, the main focus here is on studying this message during the merger.

Basically, the node density and node mobility are the two conditions considered critical in MANETs because they are the main reasons for the high likelihood of packet losses (node density) and the frequent interruptions of network connections (node mobility). While any packet drop or connection interrupt results in higher latency and signaling overhead to solve it, many networking services in different scenarios, such as the relief services in disaster scenarios, fail to tolerate latency or signaling overhead. Because a network address is the first step in any communication, there is a need for joining nodes to be assigned an address as fast as possible with low signaling overhead. The performance evaluation, therefore, focuses on three metrics, the address assignment latency, the signaling overhead and the success rates, so that the impact of network density on the assignment process as well as the impact of network node speed in all protocols can be studied. The performance of the aforementioned protocols is basically analyzed by means of simulation studies modeled in the network simulator 2 (ns2), version 2.32. This follows the IEEE 802.11 standard model in respect of Packet Error Rate (PER) to determine a random coordination. In addition, the 2.32 version uses radio propagation models which use the free space model, two-ray ground reflection mode and the shadowing model. It also uses a RTS/CTS/DATA/ACK pattern for all unicast packets and simply sends out DATA for all broadcast packets. This simulator has been extended to model the protocols studied. The implementation of MANETconf and Prophet is that of their authors; the description of Buddy protocol is that given in [52] and is relied upon in the present work.

In the merger study a successful merger in LHA is deemed to have been achieved when the broadcast message sent by a border node detects the merger, so the performance in the merger depends mainly on the successful delivery of this message to all nodes with, of course, low merger latency and low message redundancy. Thus, the performance metrics here are the success rate of the merger, the merger latency and the redundancy measured by the number of messages sent by a node for successful merger.

What follows is a special scenario to serve the analysis of the multi-hop assigning function utilized by the LHA and Buddy protocols. Then follow the main ns2 simulation setup and scenarios used to evaluate the LHA protocol with all selected protocols during the assignment function. Finally comes a description of the evaluation scenarios for the merger function in the LHA protocol.

4.1 Analysis of Multi-hop Assignment Function

Because LHA and Buddy are the only ones of the evaluated protocols to utilize a search function with the task of finding free addresses in other nodes located further than one hop away from the new node, it is important to compare the performance of the two mechanisms used by those protocols. As the search in two-hop neighbors is not sufficient as a performance metric for the protocols, a special scenario is defined in which a single node owns a number of free addresses and other nodes obtain their addresses over different hops. This scenario is implemented in the way that there are 10 stable nodes as shown in Figure 4-1. In the scenario nodes 2, 3, 4, 5 and 6 will be activated one by one every 3 seconds and get their addresses from node 0 over hops 2, 3, 4, 5 and 6 respectively, while nodes 7, 8 and 9 are set with no available free addresses to show the signaling impact of such nodes on the middle and end of the assignment path. The transmission range shown in a dotted line is set to 230 m for each node. DSDV is used as a routing protocol, while IEEE 802.11 is applied as a MAC protocol. Most results discussed here and in the following sections are shown as boxplots [85]. This type of depiction has been chosen because boxplots are more robust than classical statistics based on normal distribution, in that boxplot does not disguise the presence of outliers in the results. The method is that for each data set, a box is drawn from the first quartile to the third quartile, and the median is marked with a thick line. Additional whiskers extend from the top and bottom edges of the box towards the minimum and maximum of the data set. Data points outside the range of the box and whiskers are considered outliers and drawn separately. There is, additionally, a depiction of the mean value which is depicted in the form of a small filled square.

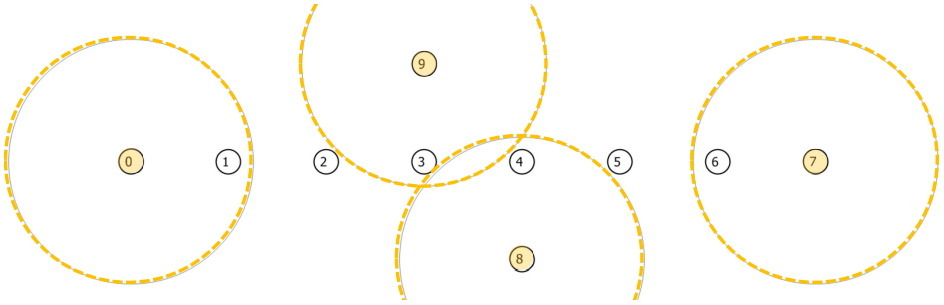


Figure 4-1: Address assignment of multi-hop Scenario

4.1.1 Assignment latency:

Figure 4-2 shows the boxplot of the latency resulting from multi-hop assignments when the number of hops varies from 2 to 6 hops; LHA and Buddy results are side by side. The scenario is repeated 10 times and each point in the figure represents the result of one of the10 trials.

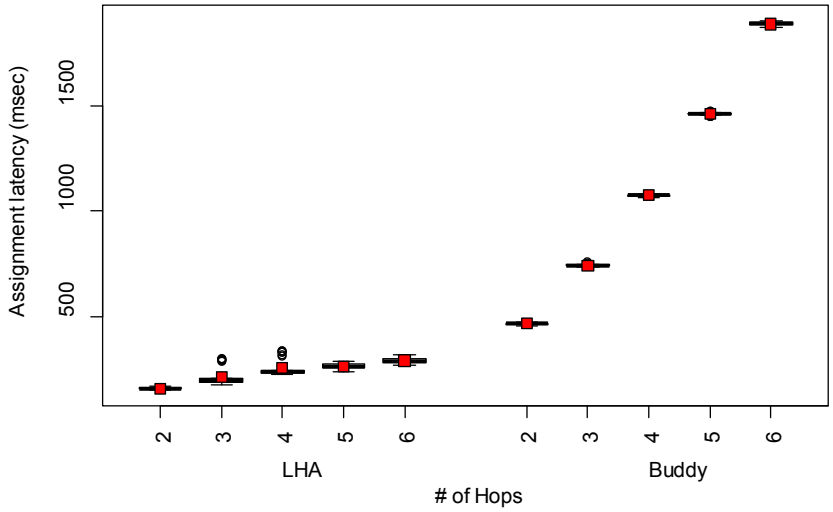


Figure 4-2: Latency impact of multi-hop assignment scenario

As the plot shows, the time required to assign an address for a new node in both protocols increases as the number of hops increases. This is normal for multi-hop assignment. However, with LHA the time required by a new node is less than the time required by Buddy: the median assignment latency employing LHA varies between 0.14 and 0.3 sec. while it varies between 0.45 and 1.86 sec. employing Buddy. The reason for this is that Buddy algorithm needs a predefined timeout (200ms as defined in the present work) to finish the search in each hop until it reaches the hop in which there is a node with a free address. This means that starting form the first hop to

the last hop, which includes a node that may have a free address, it performs subsequent search processes which in turn increase the whole searching time dramatically. This is not the case in LHA because it searches for the hop which may provide a free address by means of one search process; therefore, it uses two kinds of timeouts, a big timeout (Tm-search) used by the requester like the one used in each attempt in Buddy, and a small one (Tm-ws timeout) used by every responding node. To avoid unnecessary responses in LHA, Tm-ws is stopped upon receipt of reply messages. Because this is a special scenario with special definition (one node owns all addresses) a node with no free addresses selects[1] its timeout between 25 and 50 msec., while a node with free ones selects the timeout randomly between 0 and 25 msec. In the figure we can see that the assignment latency from hop 3 and 4 in LHA has some outliers (in some attempts, slightly lengthening the latency). This is because node 9 and 8 may try to forward the request massage if they have not overheard the reply messages sent by nodes 3 and 4 respectively.

4.1.2 Signaling Overhead:

Figure 4-3 shows the overhead resulting from multi-hop assignment when the number of hops varies between 2 and 6 hops; LHA and Buddy side by side.

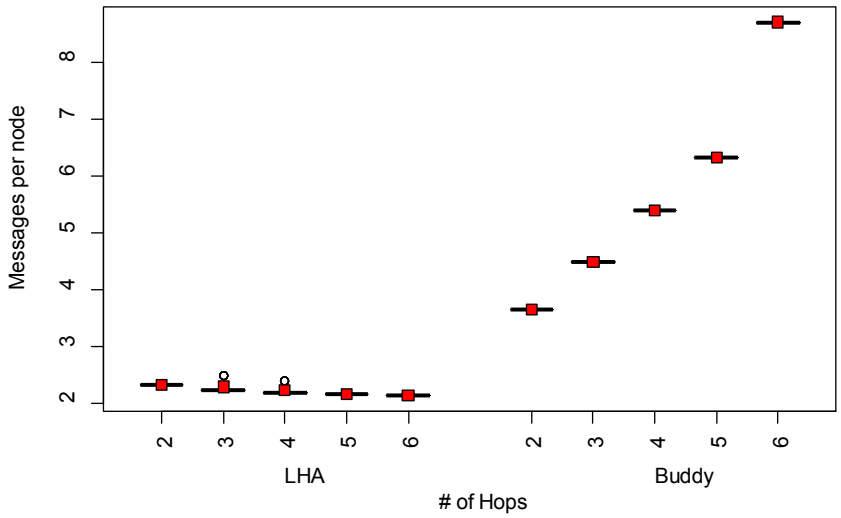


Figure 4-3: Signaling overhead of multi-hop assignment scenario

In the figure the red box (i.e. the average number of messages) shows that the average number of messages sent by each node required to join a new node deploying LHA is very low, stable and independent of the number of hops in the network (between 2 to 2.5 messages per node). On the other side, the average number of messages sent per node when Buddy is used increases sig-

[1] Random selection from the range considering the number of neighbors (high priorities for bigger number)

nificantly with an increasing number of hops. In concrete terms, the variation is between 4 and 9 messages when the number of hops increases from 2 hops to 6 hops. The reason for this is that Buddy starts a new search for each hop from the first node (new joining node) to the last hop which may include a free address. This means that the initial request message is repeated in each search within a new hop. Additionally, during the search in any hop, each configured node located within the hop and regardless of whether it has a free address or not, responds with a reply message. This increases exponentially the number of messages sent in Buddy to find a free address. In contrast, LHA requires fewer signaling messages because it utilizes a single search process. In addition due to the overhearing method applied in LHA, as already highlighted in describing LHA algorithms, LHA prevents a node from replying if there are already other responding nodes.

Because this scenario is relatively simple, using only 10 static nodes and a maximum number of 4 neighboring nodes, there are none of the packet drops or connection interrupts due to network load or node mobility that to be seen later in other scenarios. Thus, both protocols succeed in all attempts and there is no need to illustrate this. Only if successful assignment is less than 100 % is there a figure provided.

4.2 Main Scenarios of Assignment Process

To produce meaningful results, the aforementioned protocols were evaluated deploying various scenarios representing node densities (static scenarios) and mobility (mobile scenarios). Each of the scenarios represents an ad hoc network with varying placement of the nodes and varying movement abilities. The maximum simulation area of each scenario is a square 1000 m long and 1000 m wide. The transmission range of each node in all scenarios is 230 m. DSDV is used as a routing protocol, while IEEE 802.11 is applied as a MAC protocol. Because the implemented protocols are using different variables from those defined in their description, Table 4-1 presents the implementation values of the variables used in the scenarios for each protocol. In the LHA algorithm and based on RRT, a configured node selects a timeout (Tm-ws), as shown in Section 3.5.2, before the sending of reply messages (AA_Rep or AA_A_Rep). On the basis of the simulation experiments in ns2, the estimated RRT in all scenarios is set to 15ms.

In the evaluation of all the protocols applied to scenarios of varied densities and mobility, the simulation begins with one node configured with a valid IP address and operating in an ad hoc mode. In each scenario a new node joins the ad hoc network every 3 seconds and requests an IP address. This task takes goes on until the maximum number of nodes is reached (50 nodes in each scenario). If a node fails to get an IP address, it retries the request again after 200 msec for a maximum of 3 times before it decides it is to be the initial node of an ad hoc network and, thus, assigns an IP address by itself. In the simulation, the duration of each scenario is 170 sec, because the maximum number in each scenario is 50 nodes. To be sure of the results, each scenario was repeated 20 times. Here are the descriptions of the form of each scenario:

Table 4-1: the implementation values of the variables in the scenarios for every protocol

Variables	*Description*	*Values*	*Protocols*
bacon/hello	*One hope periodic message*	*2 sec*	*All Protocols*
R	*Max Retry number*	*3*	*All Protocols*
M	*Max available free addresses per node*	*3*	*LHA*
T_soll	*Solicitation timeout*	*0.9 sec*	*LHA*
T_allo	*Allocation timeout*	*2 sec*	*LHA*
T_Conf	*Configuration timeout*	*2 sec*	*LHA*
T_search	*Search timeout*	*1.6 sec*	*LHA*
RRT	*An estimated time for one hop round trip transmission in ns2*	*15 msec*	*LHA*
T_Offer	*Timeout for resend of IPAddressRequest message*	*0.2 sec*	*Buddy*
T_allo.pend	*Timeout for receive AllocatorChosen message from requester*	*0.2 sec*	*Buddy*
T_assign_ip	*Timeout for receive IPAddressAssign from the allocator where d is the hop distance from the requester to the responder*	*d*0.4 sec*	*Buddy*
T_confirm_ip	*Timeout Expected by an allocator of the search for a free address*	*d*0.4 sec*	*Buddy*
T_NR	*Timeout for neighbor reply message*	*0.2 sec*	*MANETconf*
T_MR	*Timeout for a request-reply-timer, initiator*	*1.6 sec*	*MANETconf*
T_MA	*Timeout for address allocation*	*1.7 sec*	*MANEtconf*
T_MAP	*Timeout for address pending in allocation table*	*3.4sec*	*MANETconf*
T_wr	*Wait-reply timeout*	*0.2 sec*	*Prophet*
T_ack	*Acknowledgment timeout*	*0.4 sec*	*Prophet*
N_ir	*Initiator request retry*	*2*	*MANEtconf*

- **Density scenarios:** In MANETs, the communication performance of a network is limited by the interaction between neighboring nodes, which may represent a very simple static network of chain nodes [86]. The reasons behind this are linked to the behavior of neighboring nodes, such as the sensing of the medium before the transmission of packets, and to the medium problems, such as the radio wave interference which may cause collisions at the destination nodes. In order to evaluate the performance under different network densities, 50 inactive nodes in each scenario are placed randomly within a predefined area (varying from small to big the maximum being, as mentioned above, 1000 m^2), so that a certain network density results. When the simulation starts, one node is initially activated[1] and forms the ad hoc network. Following that, a new randomly selected node is activated every 3 seconds. It is worth mentioning that no node moves during any scenario aiming at analyzing the network density. In static scenarios 4 different network densities are distinguished, namely very high, high, low and very low density. Based on these densities, the scenarios differ in the minimum

[1] Activation of a node means that the node starts joining the ad hoc network by requesting a unique IP address

number of hops between the furthest nodes in the ad hoc network. In more concrete terms, the minimum number of hops between the furthest nodes the scenario is 2, 3, 4 and 6 hops for very high, high, low, and very low density respectively. However, to satisfy the density condition, the results of the last 5 nodes joining the network were observed in each static scenario because they satisfy a certain maximum number of neighboring nodes. To clarify this, two scenarios are plotted in Figure 4-4 and ure 4-5. The first figure shows a very high density scenario, where the minimum number of hops between node 3 and 5 (the two furthest nodes in the figure) is 2. In this case, if for example node 45 enters the network it may be in density of about 33 neighboring nodes. The scenario shown in the second figure illustrates a low density scenario, where the minimum number of hops between the two furthest nodes (node 42 and 47 in the figure) is 6. In this scenario the last 5 joining nodes may be in densities between 3 to 8 modes, e.g. node 45 may have only 3 neighboring nodes (42, 43, and 15 in the figure).

- **Mobility scenarios:** A main factor affecting the communication performance of ad hoc networks is the mobility of their nodes. Therefore, it makes sense to analyze the impact of the speed of mobile nodes on the performance of the protocols. For this purpose, a scenario is built in which a joining node is a neighbor for at least one configured node in the network. The mobility pattern in each scenario follows a path built using the random waypoint[1] (RWP) model with a mean pause time of 0.5 sec between each two subsequent movements. Sample movement pattern is provided in Figure 4-6, which shows a node at every predefined instant (1 sec) randomly choosing a destination and moving towards it. On reaching the destination, the node stops for a duration defined by pause time. Then, it again selects a random destination and repeats the whole process. The speed of nodes in each simulated mobility pattern varies across the speed of a cyclist (5 m/s = 18 km/h), of a car inside a city (10 m/s = 36 km/h), of a car on an autobahn (30 m/s = 108 km/h) and of a quadcopter flight (50 m/s = 180 km/h). Movement patterns of nodes as well as initial positions of nodes in the studied scenarios remain unchanged.

[1] RWP is the most commonly used mobility model in the ad hoc networking research community as in [97].

Figure 4-4: example of very high density scenario (the minimum number of hops between the two furthest nodes is 2 hops)

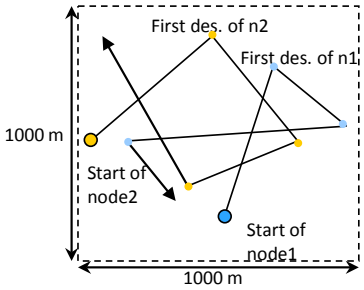


Figure 4-6: Nodes mobility pattern depending on a random way model

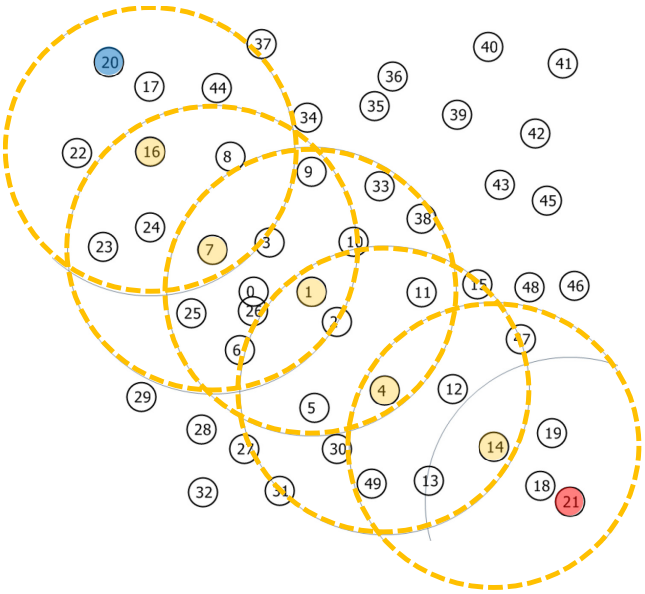


Figure 4-5: Example very low density scenario (the minimum number of hops between the two furthest nodes is 6 hops)

4.2.1 Impact of Network Density

To provide an insight into the impact of network density on the performance of the studied protocols, the assignment latency as well as protocol overhead of the address assignment process for new joining nodes in the network was investigated in detail for all predefined density scenarios. Because the overhead of broadcast messages is directly related to the broadcasting mechanisms utilized in a network, the send of this message has been handled as the send of the unicast message in the density scenarios. Basically, the study of this message is important when such messages are used periodically and when there is a need to ensure their delivery, as in protocols with centralized approaches.

- **Assignment latency:**

Figure 4-7 shows the boxplot results of the assignment latency of the last 5 joining nodes resulting from employing LHA, Buddy, Prophet and MANETConf in ad hoc networks with densities ranging from very high to low. As the figure shows, most attempts using LHA achieve low

latency under 50 msec. In the comparisons with other protocols, LHA shows stable latency in different density scenarios. As shown, LHA significantly outperforms the other protocols in high densities because it does not suffer from the high network load as is the case in the other protocols. This load is the main reason for the packet drop resulting from collisions and transmission delay due to the contentions in the wireless medium. Basically, LHA needs only a low number of messages, mainly three between the requester and the Address Agent (AA) node. Furthermore, the wait-send timeout (Tm-ws) prevents many nodes that might have received the request from sending their reply messages if there is a response meanwhile from other nodes. However, in low density, the Prophet protocol outperforms LHA because it, too, needs only a few messages in the assignment processes and, in the Prophet case, all neighboring nodes respond to the request directly. Buddy requires more latency than LHA and Prophet because Buddy requires 4 messages in every assignment process. Moreover, the requesting node needs timeout to select its allocator node from the responding ones. From the figure, one can notice that the protocol with the worst performance is, unsurprisingly, MANETConf, because the address allocation process requires permission from all nodes (global assignment decision approach), as mentioned in Section 2.3.2. The new joining node implementing MANETConf sends three messages first to select the address initiator. The address initiator in turn has to synchronize with all network nodes before assigning an IP address to the new joining node. This means that all configured nodes in the network have to participate in the assignment process, which consequent high latency. MANETConf has, moreover, a very low success rate, as shown in Figure 4-8, compared with the other protocols. Clearly, the protocol is not suitable for high node density and its performance sinks dramatically with a rising number of hops because for every additional hop the packet drops increase. In this figure it can be seen that only LHA and Prophet have a 100% success rate, which is due to the low number of messages sent between the requester and the assigning node. Buddy has more messages and less success at very high density.

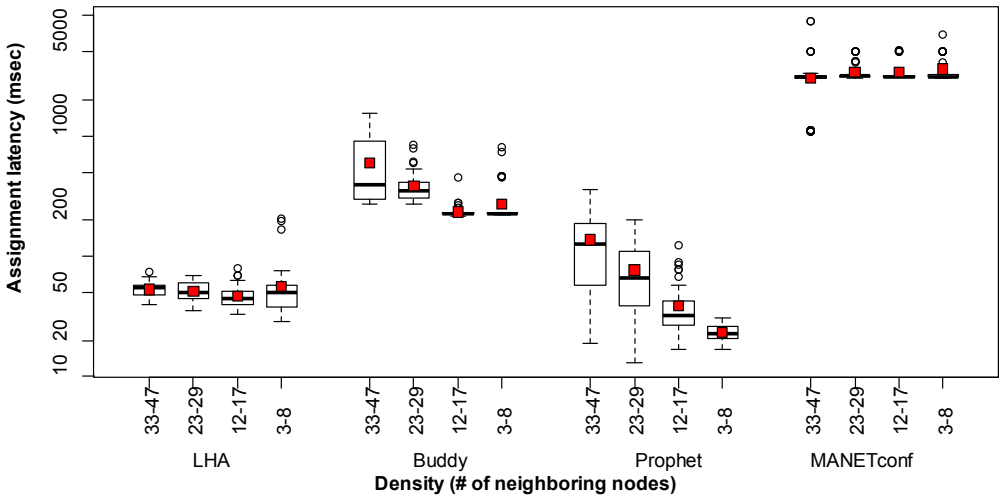


Figure 4-7: Assignment latency resulting from employing LHA, Buddy, Prophet and MANETConf in different node densities

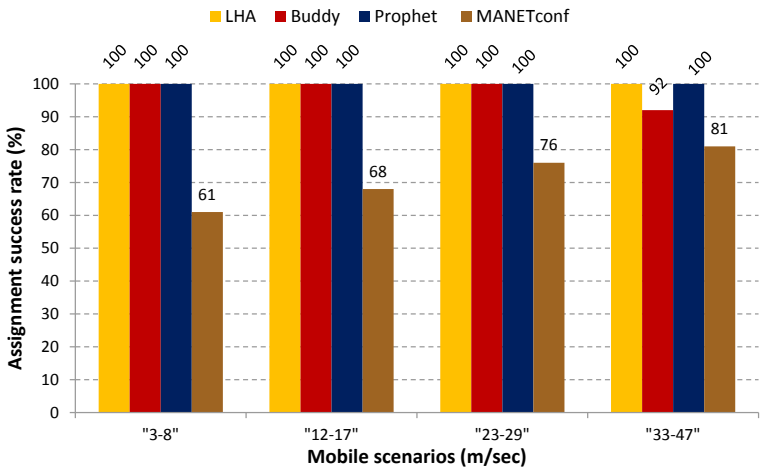


Figure 4-8: Assignment success rate of all attempts in static scenarios

- **Signaling overhead:**

Turning to signaling overhead, Figure 4-9 reveals that LHA has less signaling overhead than other protocols in all scenarios. In summary, LHA reduces the number of packets sent per time by up to 72% compared to the Buddy protocol and 61% compared to Prophet in high-density networks and by up to 58% and 44% respectively in low-density networks. Moreover, in some attempts LHA may need only 3 messages. This is because the eavesdropping mechanism used in LHA causes the neighboring nodes to discard their reply messages if other neighbors send one. Additionally, because this mechanism is based on the number of available addresses per node it allows even distribution of them across all configured nodes in the network. Basically, the efficiency of most protocols which do such distribution of available addresses, as Buddy does, is impaired because they give this task to the new node itself. This means that the new node must wait until all neighbors have sent their reply messages before it decides which neighbor to select. Of course, this mechanism consumes the transmission bandwidth and, as already shown, takes longer. In the figure, the search mechanism used by LHA and Buddy results in outliers for some attempts of very low density when responding neighbors are not able on their own to assign free addresses. The two figures, for latency and signaling overhead, make it clear that the low latency of Prophet in low-density networks has to be paid for in higher signaling overhead. This is to be expected when all neighbors of a new node have to respond directly to any request message.

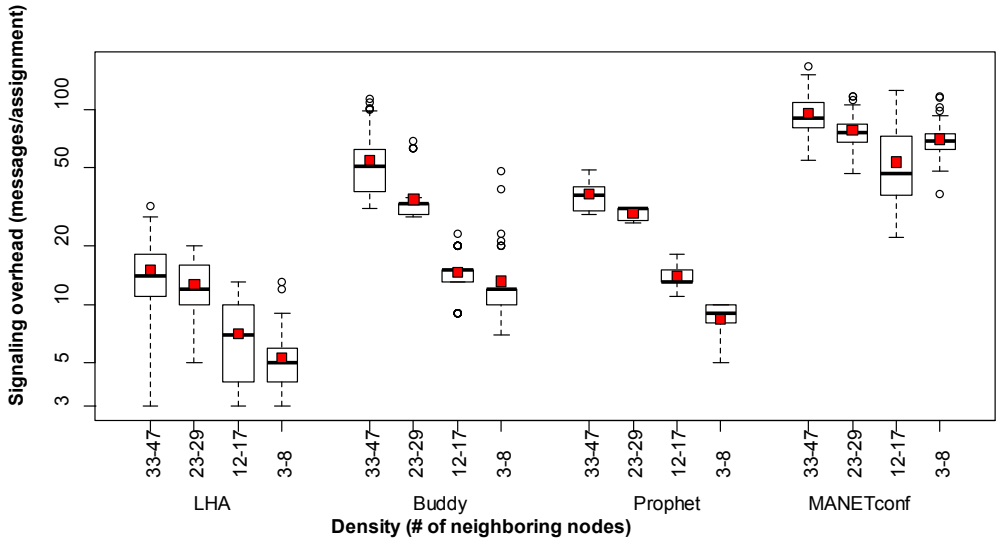


Figure 4-9: Signaling overhead resulting from employing LHA, Buddy, Prophet and MANETConf in different nodes densities

4.2.2 Impact of Node Speed

A characteristic of MANETs is that the degrees of mobility will vary, as described in [87]. Fast moving nodes indicate a network with a high degree of mobility and thus a rapidly changing topology which in turn may cause breaks in the links between communicating nodes. However, a low mobility network (one with slowly moving nodes) will have relatively stable topology. In this section, the different degrees of mobility studied and their impact on the protocols are described.

- **Assignment latency:**

It has been stated above that MANETConf shows very low performance in static scenarios. Moreover, testing of this protocol reveals a weak concept of the partitioning detection mechanism, which must be triggered in every assignment process, so that the protocol fails to work well because a detection of partitioning and merging networks has to be so frequent. For the address assignment process in mobile scenarios, therefore, the focus in this section is only on the other three protocols, where the assignment process works as long as there is a free address in the network. The boxplot in Figure 4-10 illustrates for every joining node (one node joins the network each 3 sec till the last node) the average assignment latency resulting from employing LHA, Buddy and Prophet in MANETs. The figure shows that LHA has stable assignment latency in comparison with the Buddy and Prophet protocols: 50% of all LHA latencies in the first 3 scenarios (5, 10 and 30 m/sec) are between 30 to 50 msec. This is because LHA uses a wait-send timeout to avoid packet loss due to collection and contention. Moreover, as already mentioned, the assignment process in LHA requires only a few messages. Of course, Prophet is faster in the first 2 scenarios (5 and 10 m/sec) because, like LHA, it requires few assigning messages and at the same time all nodes respond directly to the joining node. For neither of these two scenarios

does the issue of link breaks emerge, because the node speeds are not so fast. However, this is not the case at higher speeds, i.e. 30 and 50 m/sec, wherein Prophet suffers from lower success rate compared to the LHA protocol as shown in Figure 4-11. In this figure, LHA is more robust than Buddy and Prophet in all mobile scenarios and has higher reliability of successful assignment, the reason being that LHA uses broadcast messages which are more suitable than the unicast messages used in the other two protocols. As explained in the section on the LHA algorithm, the new node uses a broadcast message to inform other nodes about the new configuration. This mechanism enables LHA to avoid breaking the link between the requester and allocator; because the message can be delivered by every node hearing the message. Basically, the unicast mechanism is a big issue in highly dynamic network, affecting the other two protocols; especially the Buddy protocol with its high use of unicast messages.

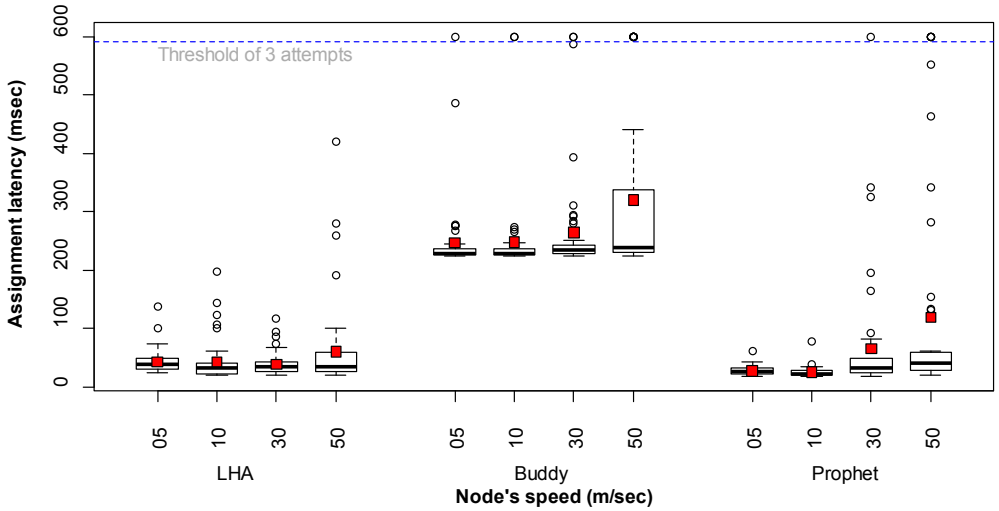


Figure 4-10: Address assignment latency resulting from employing LHA, Prophet and Buddy with different node speeds

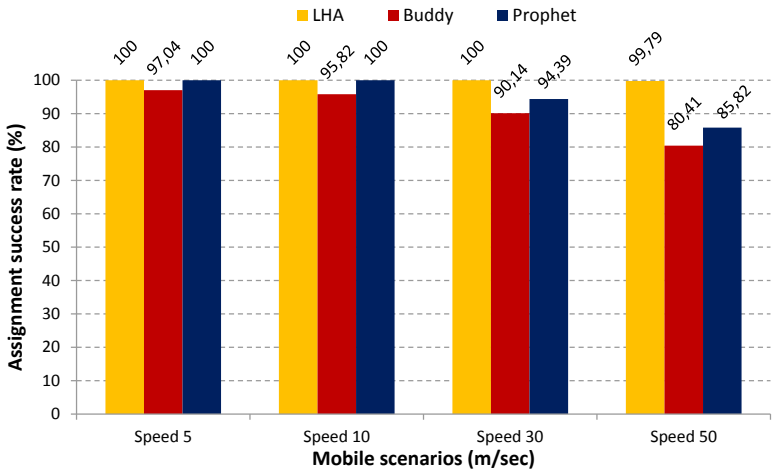


Figure 4-11: Assignment success rate for all attempts in mobility scenarios

- **Signaling overhead:**

Figure 4-12 depicts the signaling overhead of the assignment process for every node joining the network. The wide variation in the number of messages for each scenario compared to the static scenarios is due to the variety of densities in every mobile scenario. The minimum number of 3 assignment messages in LHA and Prophet and 4 messages in Buddy shown in the figure describes the case when a joining node has only one neighbor. On other hand, the maximum number of messages in Buddy and Prophet in all scenarios shown in the figure is a reflection of the approximate node density of late-joining nodes in these scenarios. For instance, in the 5m/sec scenario the maximum density is about 18 nodes, because each neighbor in Buddy and Prophet responds to any request received from a joining node. In the figure, the LHA signaling overhead in all scenarios is less than those of Prophet and Buddy because the LHA protocol uses the wait-send timeout to prevent any node from sending a response if the node hears a response from one or more neighboring nodes. In the figure, it is clear that Prophet shows less signaling overhead than does Buddy because Buddy requires more messages in each assignment process.

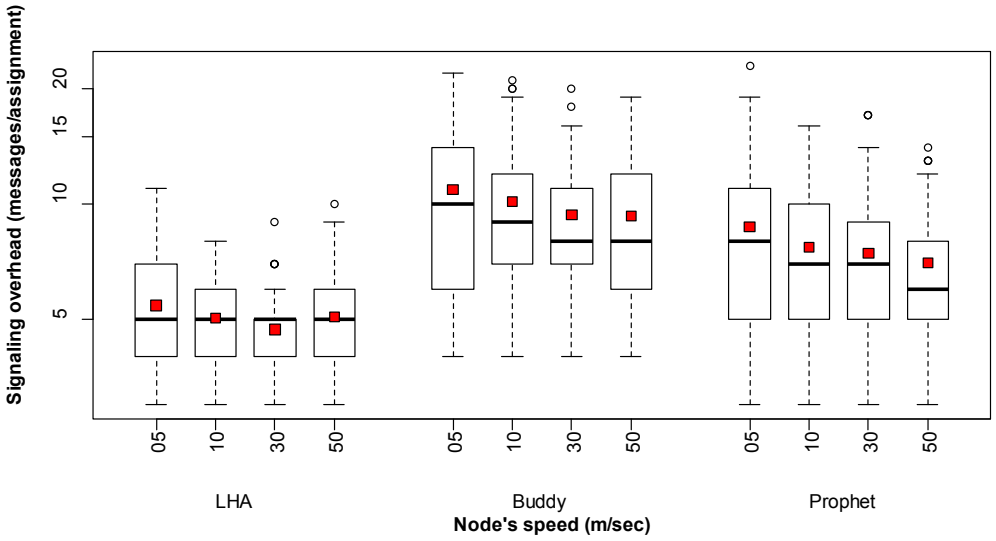


Figure 4-12: Signaling overhead resulting from employing LHA, Prophet and Buddy with different node speeds

4.3 Impact of Network Mergers

In LHA it is a task of the border nodes that detect the merger of networks to inform other nodes in the network about changes required in order to solve possible address duplications. Unlike other stateful protocols, LHA solves the conflicts by sending only one broadcast message

from a detecting node to all nodes in its network. This broadcast message is, therefore, very important during the merging of networks. Because the broadcast mechanism is not reliable against packet drops caused by such things as fading, interference or collisions, a study is here made of how, during the merger, LHA ensures the successful delivery of this message to all nodes with adequate time and message redundancy. The first focus is on a simple case in which one node detects a merger with another network. Then comes a complex situation in which several nodes detect more than one merger at the same time (simultaneous merger of more than two networks).

4.3.1 Simple Merger of Two Networks

As mentioned above, in the LHA protocol when a node detects a merger it has to send only one message that informs about the changes in its network. Basically, the merger is solved completely if every node receives the merger message. To ensure the successful delivery of the merger message, LHA uses a merger timeout in which every node analyses its neighbor's messages (this is the eavesdropping mechanism). The study now reported is, therefore, of the efficiency in terms of latency, message redundancy and merger success rate when LHA follows its own mechanism (reliable merger) or does without it (pure broadcast). In other words, if a node follows the pure broadcast a node depends only on the ongoing broadcast mechanism used in the network to deliver its message, such as the simple flooding broadcast mechanism. Because there are many reasons for message drops in wireless networks, it is important to study the merger of networks in different scenarios. Three merging scenarios have therefore been selected, representing the merging between one large and one small network, a pattern shown in Figure 4-13. The small network consists of 5 nodes and it is used only to trigger the merging function of the border nodes in the large network in which the merger message is observed. Depending on the node distribution in the larger network, the three scenarios can be described as a sparse, dense and special (clustered nodes). For example, Figure 4-13 shows a sparse scenario which indicates the case of messages dropped because of long distances between nodes. In this scenario to get full coverage from the border (node 0) a merge message has to be forwarded a minimum of 21 times. In contrast, the dense scenario reveals the problem of high collisions and contentions because of the high density of nodes, although the full coverage can be achieved with 8 forwarding episodes at a minimum.

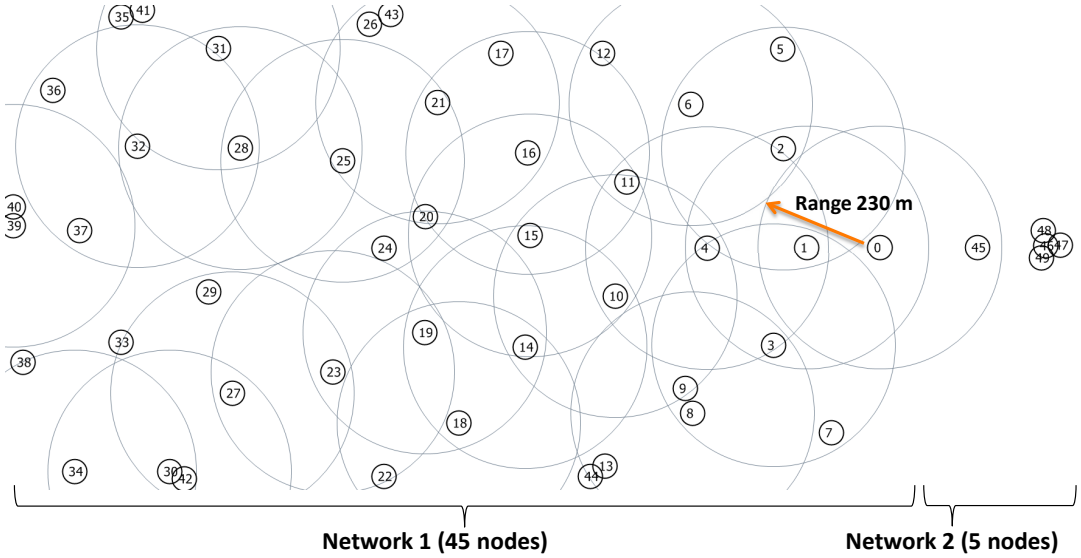


Figure 4-13: Merging networks, sparse scenario

For complex cases, a special scenario (clustered nodes) which includes all the issues from the last two scenarios as shown in Figure 4-14 is defined and studied. In this scenario, the minimum number of times the merger message is forwarded from node 0 to all nodes in the network is about 13. In addition to the "dense" and "sparse" issues, the clustered nodes scenario includes bottleneck nodes, such as nodes 2 and 3 in the figure. Because the bottleneck nodes are considered a connection node between two portions of the nodes, the dropping of the packets in one of these nodes results in a low packet delivery success rate. To show the performance of LHA merger function by using the reliable merger or pure broadcast mechanisms, we present each scenario measured by three metrics, called merger latency, message redundancy and delivery rate (i.e. full coverage success rate) as follows:

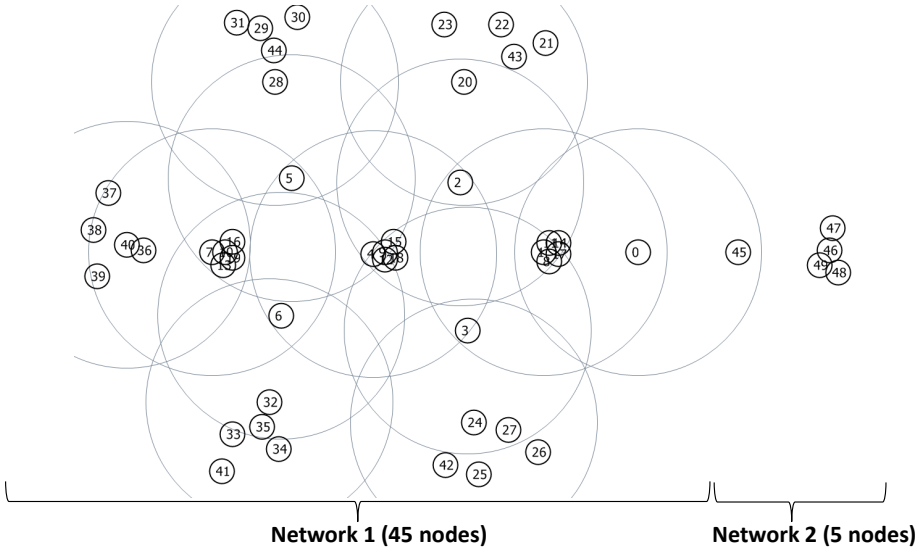


Figure 4-14: Merging networks, clustered nodes scenario

- **Success rate of merger:**

The delivery metric is the success rate of all attempts, wherein the success of an attempt indicates that the broadcast message sent during the attempt is delivered successfully to all nodes in the network which means that message has got full coverage. Figure 4-15 shows this metric for each scenario in both cases, that of using the reliable merger in LHA and that of the pure broadcast mechanism. The delivery metric shows that LHA is able to overcome the problem resulting from the broadcast mechanism. In the figure, the statistics shows that use of a pure broadcast mechanism to deliver the message to all nodes results in some unsuccessful attempts. However, the eavesdropping function in the LHA obviates this shortcoming. As can be seen, in all scenarios broadcast with the LHA eavesdropping function there is full and successful coverage. In the figure, the pure broadcast suffers mainly in sparse scenario because the forwarding requires several hops and the probability of receiving a copy of the merger message from neighboring nodes is lower than other scenarios. However, high node density in dense scenario may mitigate this problem but in turn it increases the issues of contentions and collisions between nodes. Therefore, as the figure show, the clustered nodes scenario which has moderate density of nodes shows a better success rate than the other scenarios.

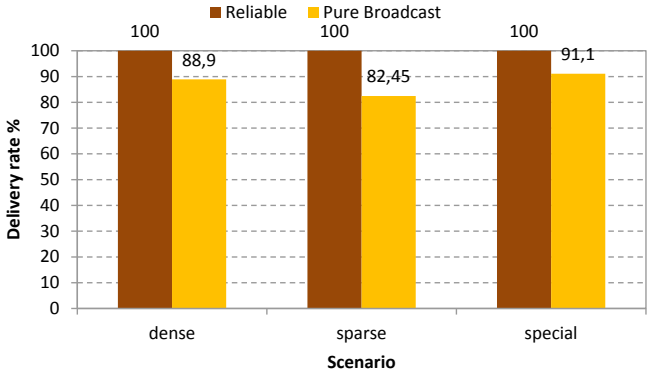


Figure 4-15: success rate of merger

- **Merger latency:**

Now follows a study of the delay in handling the merger as a function of reliable and pure broadcast mechanisms: in other words, the time required to finish the merger process successfully in both cases. Because pure broadcast is not able in all attempts to deliver a merger message successfully the latency value in Figure 4-16 indicates the latency only of successful attempts. It can be seen that the maximum average time needed to inform all nodes in the network by using pure broadcast mechanism in all scenarios is lower than that in the reliable mechanism and that the latency is less than 0.5 sec in all scenarios. Basically, the reliable mechanism used in LHA merging function results in higher latency due to the waiting state needed for reliable merging, i.e, for success in all attempts as presented in Figure 4-15 above. As explained in Section 3.6.2, during this state each node forwarding any merger message (S_Merge or H_Merge) must detect and resend the message if the first forwarding has failed to achieve full coverage. Of course the time needed in a dense scenario in some cases is higher because the number of nodes trying to forward or resend the dropped messages is higher and this leads to increased transmission time. This effect is less in the sparse scenario but, on other hand, there are more outliers indicating resend function due to dropped packets in this scenario because of the many hops and few neighboring nodes. In this scenario 90% of latencies are between 0.4 and 0.5 second because the minimum number of forwarding episodes required to get full coverage is bigger than that in the other two scenarios. In clustered nodes scenario the outliers is due to the bottleneck nodes but the most latencies in this scenario is lower than that in sparse scenario because it needs less number of hops for the forwarding.

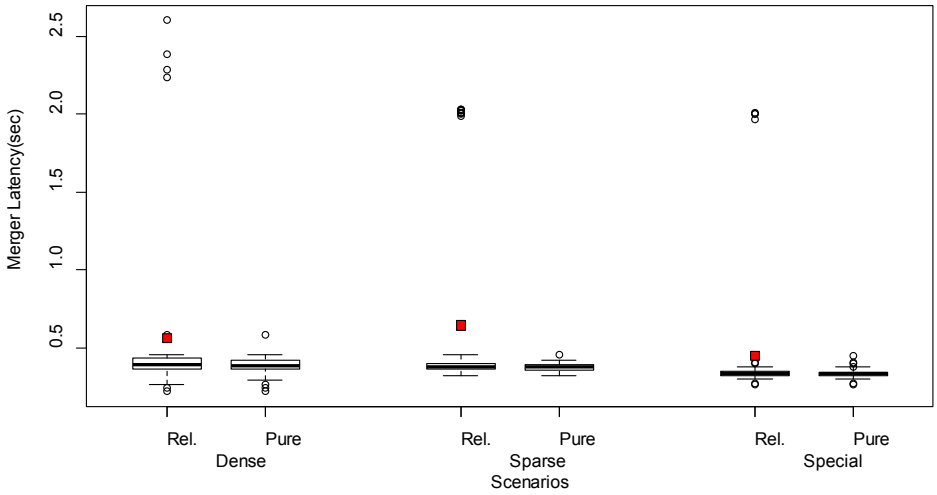


Figure 4-16: Latency of successful networks merger in different scenarios

- **Message redundancy:**

Figure 4-17 illustrates the message redundancy for each scenario, measured by number of messages sent per node. It is clear that in successful pure broadcasting attempts the merger message will be sent once by every node. However, to get full success in all attempts, the reliable merger of LHA allows a node to resend a message if there is a message drop. Therefore, more messages are sent per node in a dense scenario than in the other two scenarios, where many neighboring nodes try to resend the dropped message to the same destination. In contrast, the sparse scenario does not suffer from this case because of the low number of neighboring nodes distributed over big number of hops. Unsurprisingly, the results for the clustered nodes scenario show an average value between the two first scenarios because it has less node density than the dense scenario and more than the sparse scenario.

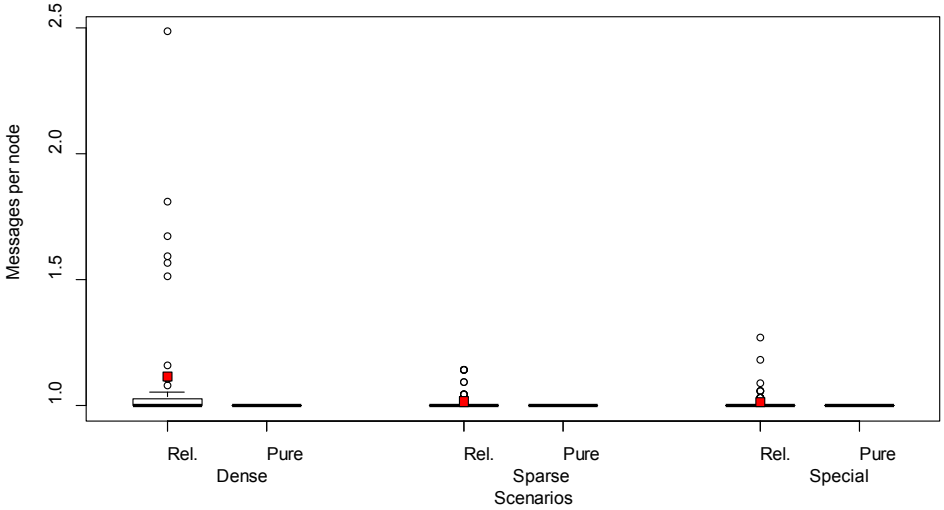


Figure 4-17: Message redundancy of successful networks merger in different scenarios

4.3.2 Simultaneous Merger of More Than Two Networks

To evaluate LHA under critical conditions a case is studied where more than two networks merge simultaneously. Moreover, to prove that the merging algorithm of LHA is fully distributed in that each node is able to solve the merger without a conflict arising with other nodes, a scenario is studied in which more than one node detects and solves the merger. Figure 4-18 shows a scenario where three networks (A, B and C) merge simultaneously. In the figure the distance between the border nodes in both A and B is equal to that between those in B and C. In this scenario, network B remains stable during the simulation while A and B networks are approaching B at the same speed, causing simultaneous merging of the two different networks (A and C) with the B network. Because nodes 9, 10 and 11 of network A are aligned vertically and are at the same level as nodes 27, 28 and 29 respectively, the connection time may be identical, which means one or all of them may detect and solve the merger. This situation is one of the big issues in any distributed auto-configuration function which allows every node to be responsible for solving possible address conflicts in the network. In LHA this case is competently solved, as described in Section 3.6.6, where each node is able to inform other nodes in the network of the required changes without giving rise to any conflicts. By means of this scenario, LHA is evaluated on three main metrics: merger latency, signaling cost and success rate of the merging.

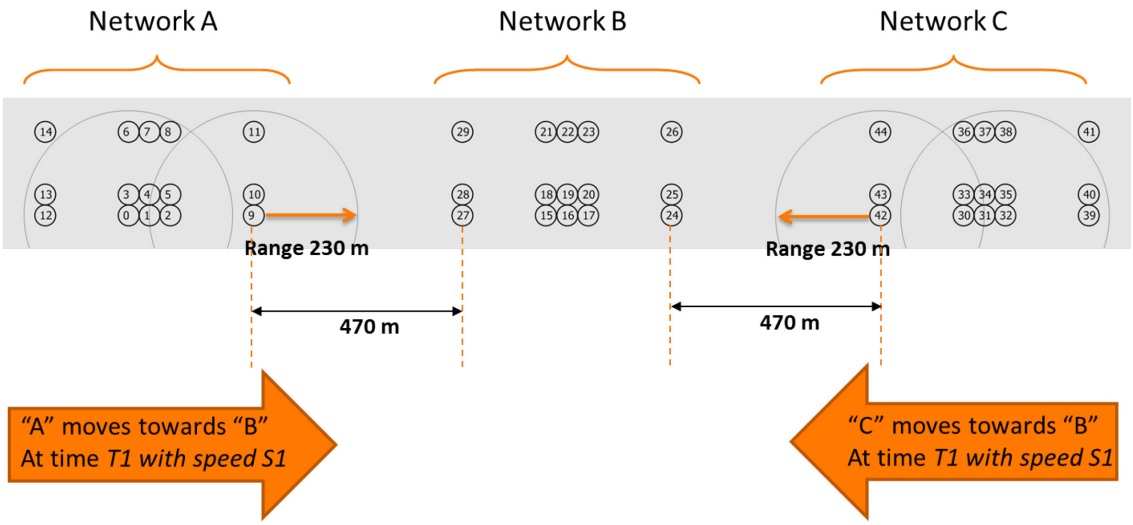


Figure 4-18: Example of simultaneous merger of 3 networks

- **Message Redundancy**

In LHA every node is able to detect and solve the merger by sending a corresponding merger message (S_Merge or H_Merge). In scenario studied, the merging networks have different HierIDs. So when a node detects a merger it will send an S_Merge message to its network. Fig-

ure 4-19 shows the number of messages sent per node in each network. In the figure, for about 50% of the nodes in networks A and C only two messages are shown as basis sent because the nodes in each network will receive two different S_Merge messages (each including different configuration data). One of these is produced when A (or, as the case, may be, C) merges with the relevant half of the nodes in network B. Simultaneously, the other network is merging with the other half of the nodes in B. The other S-Merge message goes out when both these two new networks merge, see Section 3.6.6 for more details. The reason why some nodes do not send the message is that the optimized broadcast mechanism [79] used in this scenario reduces the number of broadcast messages if all neighbors receive a copy. In contrast, the figure reveals that the number of messages sent by 50% of nodes in network B is between 2 and 4 messages. This is because many nodes in B detect the merger with A and, at the same time, the merger with C and they try to inform other nodes about that. The resulting high collision and contention among nodes of network B then prevents some nodes from receiving the message. When the senders detect the dropping they will try to resend the message. In some cases, this message may be sent up to 8 times.

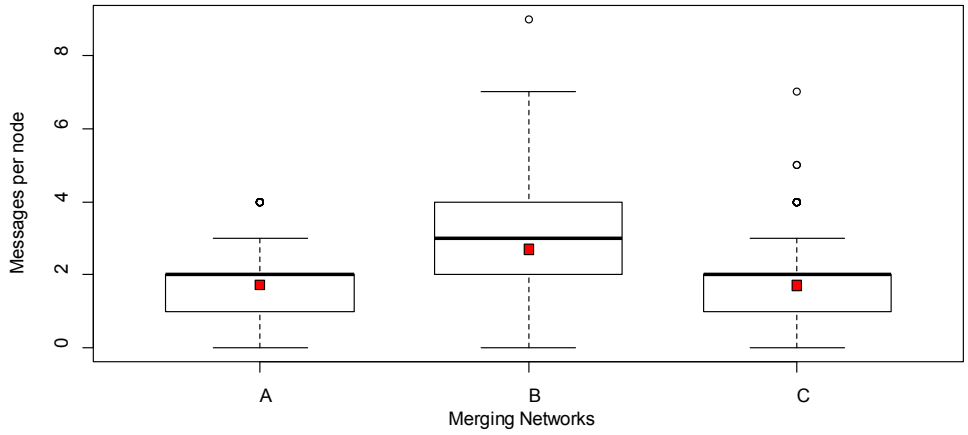


Figure 4-19: Number of messages sent per node in case of simultaneous merging of 3 networks

- **Merger latency**

Basically, the number of messages needed by the detecting nodes in network B for the first step of the merger results in high latency to achieve the second step. Figure 4-20 shows that the mean latency to finish the second merger among the nodes in network B is about 4 seconds and the maximum latency for 90% of all attempts is about 4.5 seconds. This is usual, because some of B nodes merge first with A nodes and others with C nodes. So, after the merger timeout (2 sec) each part of nodes from B detects that there is another merger among them and tries to solve the merger. This is the reason why the nodes in network A and C require bigger latency than the nodes of B alone. In the figure the minimum latencies of A and C are above 4 seconds. However, the outliers show the cases when a node in some attempts does not receive the merger message from its neighbors. In this case, the neighbors detect the old state of this node when it sends a

Beacon message. Therefore, the message may be sent twice or more to solve the problem and get full success in the merging.

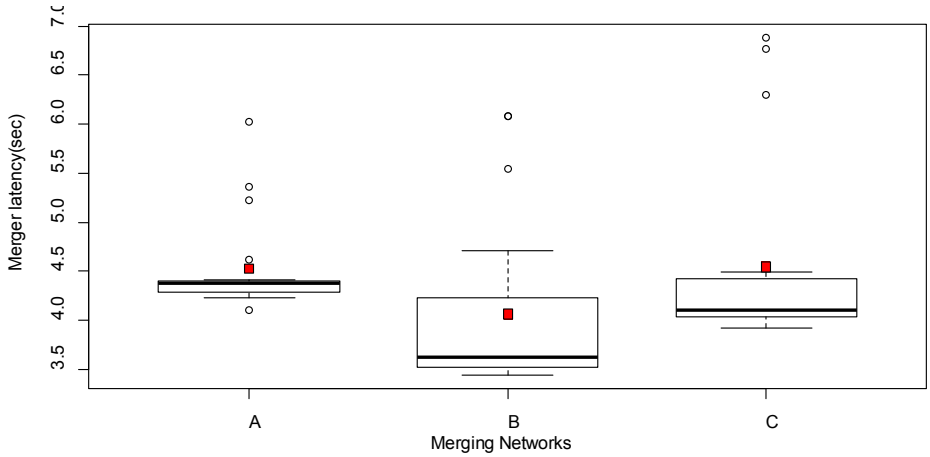


Figure 4-20: Merger latency in case of simultaneous merging of 3 networks

- **Success rate of merger:**

Figure 4-21 illustrates the success rate of all attempts for the merging of the three networks (A, B and C), at the end of which every node has achieved the required changes. This means that every node from the merged networks (A, B and C) has successfully selected a new Hash number and MN as described in the LHA merging algorithm.

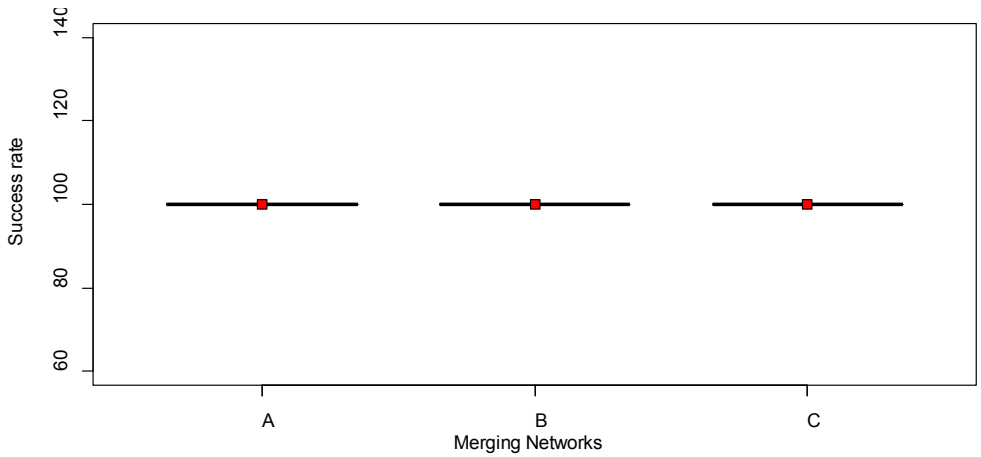


Figure 4-21: Success rate in case of simultaneous merging of 3 networks

Chapter 5 Conclusion & Future Work

In this doctoral thesis, multiple research problems related to the auto-configuration of network addresses in MANETs have been investigated; mainly, the auto-configuration problems in case of the node joining/departure and networks partitioning/merging. To explain the drawbacks of solutions available to date, a survey on current addressing protocols in traditional network (Internet network) and ad hoc networks has been presented. A discussion of the available classification of auto-configuration protocols in MANETs has followed. Because stateful auto-configuration protocols ensure by their concept the unique assignment of network addresses, the main focus in this work has been on studying those protocols. The existing stateful approaches have been classified according to the assignment process applied, into two major categories (centralized and distributed). The analytical comparison of stateful approaches has shown that the distributed ones are efficient in signaling cost and assignment latency. However, the distributed protocols suffer by virtue of their concept of address management, such as address reuse. Moreover, the new classification of possible address conflict resolution mechanisms (in cases of merger) has shown that all available protocols in MANETs suffer from high signaling overhead in attempting to solve the conflicts in the merger scenarios. This has triggered the development of an efficient stateful address auto-configuration, called LHA, to take place as a *local first* decision of any distributed mechanism.

In LHA, each node is able to allocate and assign a free address to a joining node, employing a new function which has been designed to distribute the available address space evenly across configured nodes. Moreover, the basic idea of dividing the Host ID of a node address into HierID and HHiD gives LHA the basis for efficient solutions of network merger and partition. In contrast to most protocols developed to date, LHA presents an efficient and robust solution for the merging of networks, in that with LHA every node detecting a merger is able to solve the address conflicts resulting from the merger by sending only one broadcast message. By the use of the LHA concept, the change of all network addresses is achieved uniformly, which in turn provides routing protocols in MANETs with a seamless way of updating the routing table. Thus, possible interruptions of ongoing communications due to network address changes can be avoided. Moreover, the merger algorithm is able to handle simultaneous merging of more than two networks. In all simulation attempts, the results have shown successful merging within a satisfactory time.

In the present work, the simulator ns2 has been used to evaluate LHA vis à vis three counterpart stateful protocols with distributed approaches; namely the Buddy, MANETConf and Prophet protocols. Although the address assignment process of stateful protocols ensures the unique assignment of network addresses in network, if the assignment of a free address is not completed successfully there may be address conflicts in the network. A successful process should, of course, come at a low cost in time and signaling overhead. The evaluation results have shown

that LHA outperforms Buddy, Prophet and MANETConf for both signaling cost and assignment success rate in all scenarios studied. Comparing the assignment latency results has shown that LHA is faster than the other protocols in such critical scenarios as high-density networks and high-speed mobile scenarios. Specifically, when the assignment latency data for LHA, Buddy and Prophet are compared, the results have shown that LHA is up to 86% faster than Buddy and 57% faster than Prophet in high-density networks. In low-density networks, LHA is up to 77% faster than Buddy but Prophet outperforms LHA, where it is 54% faster. However, this outperformance by Prophet in low-density networks has to be paid for in high signaling overhead. The comparison has shown that LHA reduces the number of packets sent per time by up to 72% compared to Buddy and 61% compared to Prophet in high-density networks and in low-density networks, by up to 58% and 44% respectively.

For the mobile scenarios, a study of MANETconf has been omitted because of the features of its partitioning/merging function and its low performance in static scenarios as mentioned in Section 4.2.2. In all scenarios when there are rapidly moving mobile nodes, LHA has performed with signaling cost significantly better than Buddy and Prophet. The average number of messages transmitted by LHA is approximately 50% fewer than the number in Buddy and 30% fewer than in Prophet. With regard to average address assignment latency, LHA is up to 80% faster than Buddy in all scenarios and it is up to 40 % faster than Prophet in high-speed scenarios. Although Prophet is faster than LHA in low-speed scenarios, the latency differences in those scenarios are small, with a satisfactory time of under 50 msec. With respect to successful assignment process, LHA has shown better performance in all mobile scenarios than that of Prophet or Buddy. All these results in mobile scenarios taken together prove that it is more suitable overall for such scenarios than are the other protocols.

Because LHA, contrary to other protocols, solves the conflicts during the merger by sending only one broadcast message from a detecting node to all nodes in its network, the merger aspects has been studied only for LHA. The study has focused on terms (latency, message redundancy and merger success rate) when LHA follows its own mechanism (reliable merger) or does without it (pure broadcast). The results of merger scenarios have shown that the reliable merger function of LHA overcomes the low success rate resulting from the pure broadcast mechanism. However, this has to be paid for higher redundancy and latency. To address the simultaneous merging of more than two different networks, which is one of big challenges in MANETs, a critical scenario in which different nodes from each network detect the merger and try to solve them has been studied. The results have shown that LHA copes with these multiple detections successfully and manages the simultaneous mergers in adequate time (90% of results between 3.5 and 5.5 seconds) with merely normal redundancy (the averages are 2 to 3 messages per node).

5.1 LHA Features

- **Unique and rapid address assignment:** LHA utilizes a distribution function based on *local first* approach with fast search mechanism.

- **Fast and flexible merging function:** because of the hierarchical aspects each node in LHA detects and defines the required changes in the network. This in turn enables LHA to handle different kinds of merger, such as the simultaneous merger of more than two networks.
- **Minimum protocol overhead:**
  - This is achieved in the assignment process because LHA prevents the neighboring nodes of a new node from responding if they hear a response from other neighbors.
  - In the merger process (for resolving address duplications): LHA utilizes a uniform change which is applied in a distributed manner. In this way any node detecting a merger is able to solve any possible conflicts by sending only a single broadcast message per a merger.
- **Reliable address reuse:** by means of the LHA assignment function every configured node is able to detect easily the successors of any predecessor. In this way, the responsibility for managing the addresses of missing nodes is given to an appropriate node when the predecessor is not available in the network.
- **Robust against dropped control packets**: LHA uses flexible updating mechanism by adding its parameters to the Beacon message; for example, if some nodes do not receive the information from a new node due to the dropping of the New_Node message the information of this new node may be obtained from Beacon of the neighboring nodes. Moreover, for critical broadcast messages, such as the merger messages, LHA utilizes a variety of timers and overhearing mechanisms to ensure high delivery.
- **Lightweight algorithm for partitioning network**: on the bases of whether free addresses are available in the network, LHA decides to the change the partition configuration. This means that if there are available addresses in a network there is no need to make any changes.

5.2 Future Work

LHA has, in fact, been implemented using a Linux testbed as presented in [71] though the implementation was realized in the prior publication without wait-send timeouts. The next step would, therefore, be to test the functions of LHA under these constraints.

Because LHA employs a uniform method to modify the address data of all nodes in the address tables, it may be possible for the routing protocols to make use of this method to modify the routing tables without affecting ongoing connections. However, this requires more research to define how to modify any routing protocols which depend on such tables. A possible way might be that these protocols provide their tables to LHA which will update them as necessary.

Further possible work is to use the LHA principle to enhance and solve the IP address issues for the connection of MANETs to the Internet. As known, NAT [88] is used to extend the life of IPv4 by translating a set of private IP addresses into a public IP addresses to reach the global Internet and vice versa. However, the technique has scalability problems because many hosts wish to assign private addresses, and more latency is associated with processing each packet crossing through the NAT, and so on. Because LHA uses a multi-hierarchical structure it is not required to pick up an address for each node of a MANET but only to pick a HierID for an address hierarchy. This in turn enables each node to be assigned a unique address locally which can be used directly in the Internet.

Finally, in auto-configuration schemes the security issue (such as address spoofing attack and Sybil attack which may lead to the exhaustion of address space or to address conflict) represents another research area: the development of a secure auto-configuration protocol. Many researchers have tried to improve their protocols by adding a security function as is the case in Reshmi *et al*. [89]. Thus, security in LHA represents another aspect which requires additional future work to enable development of a new secure scheme based on the logical structure.

Bibliography

[1] R. Hekmat, *Ad-hoc Networks: Fundamental Properties and Network Topologies*. Netherland: Springer, 2006.

[2] S. K. Sarkar, T. G. Basavaraju, and C. Puttamadappa, *Ad Hoc Mobile Wireless Networks: Principles, Protocols, and Applications*, T. &. F. Group, Ed. New York, USA: Auerbach Publications, 2008.

[3] Defense Advanced Research Projects Agency (DARPA). [Online]. http://www.darpa.mil/index.html

[4] R. Bolt, L. Beranek, and R. Newman, "A HISTORY OF THE ARPANET: The First Decade," DARPA DTIC File 4799, 1981.

[5] BMW-Group. (2011) TALKING CARS FOR LESS CONGESTION - THE FUTURE OF TELEMATICS. [Online]. https://www.press.bmwgroup.com/

[6] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, p. 2787–2805, Jun. 2010.

[7] S. Basagni, M. Conti, S. Giordano, and I. Stojmenovic, *Mobile Ad Hoc Networking*, 0471373133rd ed. IEEE, 2004.

[8] C. Perkins. (1998, Nov.) "Mobile Ad Hoc Networking Terminology"Internet Engineering Task (IETF) Internet Draft. [Online]. http://tools.ietf.org/id/draft-ietf-manet-term-01.txt

[9] J. Lee, G. Kim, and S. Park, "Optimum UDP packet sizes in ad hoc networks," in *Workshop on, High Performance Switching and Routing, Merging Optical and IP Technologies*, Kobe, Japan, 2002, pp. 214-218.

[10] S. Corson and J. Macker. (1999, Jan.) Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations, IETF, RFC 2501. [Online]. http://tools.ietf.org/html/rfc2501

[11] D. B. Walker . (1997, Aug.) PROCEEDINGS OF THE THIRTY-NINTH INTERNET ENGINEERING TASK FORCE. [Online]. http://www.ietf.org/proceedings/39/

[12] J. Seitz, M. Debes, M. Heubach, and R. Tosse, *Digitale Sprach- und Datenkommunikation: Netze - Protokolle - Vermittlung*, 1st ed. München, Germany: Hanser Wirtschaft, 2007.

[13] E. Baccelli and M. Townsley. (2010, Sep.) IP Addressing Model in Ad Hoc Networks, IETF RFC(5889). [Online]. http://tools.ietf.org/pdf/rfc5889.pdf

[14] IEEE Standard 802.11. (1999, Aug.) Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

[15] R. Droms. (1997, Mar.) Dynamic Host Configuration Protocol, IETF, RFC 2131. [Online]. http://www.ietf.org/rfc/rfc2131.txt

[16] S. Cheshire, B. Aboba, and E. Guttman. (2005, May) Dynamic Configuration of IPv4 Link-Local Addresses. [Online]. https://tools.ietf.org/html/rfc3927

[17] S. Thomson and T. Narten. (1998, Dec.) IPv6 Stateless Address Autoconfiguration, IETF, RFC 2462. [Online]. https://datatracker.ietf.org/doc/rfc2462/

[18] X. Chu, J. Liu, and Y. Sun, "Address Allocation Mechanisms for Mobile Ad Hoc Networks," in *Guide to Wireless Ad Hoc Networks*, L. Limited, Ed. London, UK: Springer, 2009, ch. 14, pp. 333-354.

[19] B. Croft and J. Gilmore. (1985, Sep.) BOOTSTRAP PROTOCOL (BOOTP), IETF, RFC 951. [Online]. http://www.ietf.org/rfc/rfc951.txt

[20] R. Droms, J. Bound, B. Volz, T. Lemon, and C. Perkins. (2003, Jul.) Dynamic Host Configuration Protocol for IPv6 (DHCPv6), IETF, RFC 3315. [Online]. http://www.ietf.org/rfc/rfc3315.txt

[21] E. Guttman, "Autoconfiguration for IP Networking: Enabling Local Communication," *IEEE INTERNET COMPUT*, vol. 5, no. 3, pp. 81-86, May 2001.

[22] A. Williams. (2002) "Requirements for Automatic Configuration of IP Hosts"Internet Engineering Task (IETF) Internet Draft. [Online]. http://www.watersprings.org/pub/id/draft-ietf-zeroconf-reqts-12.txt

[23] R. Hinden and S. Deering. (2006, Feb.) IP Version 6 Addressing Architecture, IETF, RFC 4291. [Online]. http://tools.ietf.org/html/rfc4291

[24] Cisco. (1997, May) DuplicateMACAddresses on Cisco 3600 Series. [Online]. http://www.cisco.com/c/en/us/support/docs/field-notices/misc/7.html

[25] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. (2007, Sep.) Neighbor Discovery for IP version 6 (IPv6). [Online]. https://tools.ietf.org/html/rfc4861

[26] N. I. C. Wangi, R. V. Prasad, M. Jacobsson, and I. Nienegeers, "Address autoconfiguration in wireless ad hoc networks: protocols and techniques," *IEEE Wireless Communications*, vol. 15, no. 1, pp. 70-80, Feb. 2008.

[27] K. Weniger and M. Zitterbart, "Address Autoconfiguration in Mobile Ad Hoc Networks: Current Approaches and Future Directions," *IEEE Network* , vol. 18, no. 4, pp. 6-11, Jul. 2004.

[28] K. Manousakis and J. S. Baras, "Network and Domain Autoconfiguration: A Unified Approach for Large Dynamic Networks," *IEEE Communication Magazine*, vol. 43, no. 8, pp. 78-85, Aug. 2005.

[29] L. J. G. Villalba, J. G. Matesanz, A. L. S. Orozco, and J. D. M. Díaz, "Auto-Configuration Protocols in Mobile Ad Hoc Networks," *OPEN ACCESS Journal Sensors*, vol. 11, pp. 3652-3666, Mar. 2011.

[30] M. B. Mutanga, P. Mudali, and M. O. Adigun, "Towards auto-configuring routing protocols for wireless ad-hoc networks," *International Journal of Computer Engineering Research (IJCER), Academic Journals*, vol. 2, no. 2, pp. 19-27, Mar. 2011.

[31] A. Munjal and Y. N. Singh, "Review of stateful address auto configuration protocols in MANETs," *Journal Ad Hoc Networks, Elsevier*, vol. 33, no. C, pp. 257-268, Oct. 2015.

[32] S. Bansal and P. K. Gaur, "Review of Auto-configuration Protocols for WANETs for IPv4 and IPv6 networking," *International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE)*, vol. 4, no. 3, pp. 460-467, Mar. 2015.

[33] C. E. Perkins, E. M. Royer, and S. R. Das. (2001, Nov.) "IP address autoconfiguration for ad hoc networks"Internet Engineering Task (IETF) Internet Draft. [Online]. https://tools.ietf.org/html/draft-perkins-manet-autoconf-01

[34] M. Fazio, M. Villari, and A. Puliafito, "AIPAC: Automatic IP address configuration in mobile ad hoc networks," *Elsevie rComputer Communications (COMCOM)*, vol. 29, no. 8, pp. 1189-1200, May 2006.

[35] N. H. Vaidya, "Weak Duplicate Address Detection in Mobile Ad Hoc Networks," in *ACM MobiHoc*, Lausanne, Switzerland, 2002, pp. 206-216.

[36] K. Weniger, "Passive Duplicate Address Detection in Mobile Ad Hoc Networks," in *Proc. IEEE WCNC*, New Orleans, LA, USA, 2003, pp. 1504-1509.

[37] C. Davids, G. Ormazabal, and R. State, "Real-Time Communications: Topics for Research and Methods of Collaboration," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 112-115, Jul. 2014.

[38] M. Günes, "Routing und Adressierung in mobilen multi-hop Ad-hoc-Netzen," Ph.D., Fakultät für Mathematik, Informatik und Naturwissenschaften, Technischen Hochschule Aachen, Deutschland, 2004.

[39] S. Nesargi and R. Prakash, "MANETconf: Configuration of Hosts in a Mobile Ad Hoc Network," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, New York, NY, 2002, pp. 1059-1068.

[40] M. Mohsin and R. Prakash, "IP Address Assignment in a Mobile Ad-hoc Network," in *In Proceedings of IEEE MILCOM*, Anaheim, USA, 2002, pp. 856-861.

[41] K. Wenige, "PACMAN: Passive Autoconfiguration for Mobile Ad Hoc Networks," *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS*, vol. 23, no. 3, pp. 507-519, Mar. 2005.

[42] S. R. Hussain, S. Saha, and A. Rahman, "An Efficient and Scalable Address Autoconfiguration in Mobile Ad Hoc Networks," in *8th International Conference (ADHOC-NOW)*, Murcia, Spain, September, 2009, pp. 22-25.

[43] S. R. Hussain, S. Saha, and A. Rahman, "SAAMAN: Scalable Address Autoconfiguration in Mobile Ad Hoc Networks," *Journal of Network and Systems Management ,Springer Science+Business Media*, vol. 19, no. 3, pp. 394-426, Sep. 2011.

[44] T. R. Reshmi and K. Murugan, "Filter-based address autoconfiguration protocol (FAACP) for duplicate address detection and recovery in MANETs," *Computing, Springer-Verlag Wien*, vol. 97, no. 3, pp. 309-331, Mar. 2015.

[45] Y. Sun and E. M. Belding-Royer, "Dynamic Address Configuration in Mobile Ad hoc Networks," Computer Science Department, University of California Santa Barbara (UCSB), California, USA, Technical Report 2003-11, March 2003.

[46] M. Nazeeruddin, G. Parr, and B. Scotney, "DHAPM: A New Host Auto-configuration Protocol for Highly Dynamic MANETs," *Journal of Network and Systems Management*, vol. 14, no. 3, pp. 441-

475, Sep. 2006.

[47] M. Kim, M. Kumar, and B. Shirazi, "A Lightweight Scheme for Auto-configuration in Mobile Ad Hoc Networks," in *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, Denver, Colorado, USA, April 2005.

[48] M. Kim, M. Kumar, and B. Shirazi, "An Integrated Scheme for Address Assignment and Service Location in Pervasive Environments," *Embedded and Ubiquitous Computing – EUC 2005, the series Lecture Notes in Computer Science*, vol. 3824, pp. 967-976, 2005.

[49] T. Xu and J. Wu, "Quorum Based IP Address Autoconfiguration in Mobile Ad Hoc Networks," in *27th International Conference on Distributed Computing Systems Workshops (ICDCS 2007 Workshops)*, Toronto, Ontario, Canada, 2007.

[50] U. Joung and D. Kim, "2-Level Hierarchical Cluster-Based Address Auto-configuration Technique in Mobile Ad-Hoc Networks," *Ubiquitous Intelligence and Computing, the series Lecture Notes in Computer Science*, vol. 4611, pp. 309-320, 2007.

[51] M. Mohsin and R. Prakash, "IP ADDRESS ASSIGNMENT IN A MOBILE AD HOC NETWORK," in *in Proceeding of IEEE Military Communications Conference (MILCOM 2002)*, Anaheim, CA, USA , 2002, pp. 856-861.

[52] M. Thoppian and R. Prakash, "A Distributed Protocol for Dynamic Address Assignment in Mobile Ad Hoc Networks," *IEEE Transactions on Mobile Computing*, vol. 5, no. 1, pp. 4-19, Jan. 2006.

[53] Y.-S. Chen, T.-H. Lin, and S.-M. Lin, "RAA: a ring-based address autoconfiguration protocol in mobile ad hoc networks," *Wireless Personal Communications Journal*, vol. 43, no. 2, pp. 549-571, Apr. 2007.

[54] L. J. G. Villalba, J. G. Matesanz, A. L. S. Orozco, and J. D. M. Díaz, "Distributed Dynamic Host Configuration Protocol (D2HCP)," *sensors*, vol. 11, pp. 4438-4461, Apr. 2011.

[55] L. J. G. Villalba, A. L. S. Orozco, J. G. Matesanz, and T. H. Kim, "E-D2HCP: enhanced distributed dynamic host configuration protocol," *Computing, Springer-Verlag Wien*, vol. 96, no. 9, pp. 777-791, Sep. 2014.

[56] K. Knowlton, "Fast Storage Allocator," *Communications of the ACM*, vol. 8, no. 10, pp. 623-624, 1965.

[57] J. L. Peterson and T. A. Norman , "Buddy Systems," *Communications of the ACM*, vol. 20, no. 6, pp.

421-431, 1977.

[58] T. Clausen and P. Jacquet. (2003, Oct.) Optimized Link State Routing Protocol (OLSR), IETF, RFC 3626. [Online]. http://tools.ietf.org/html/rfc3626#section-3

[59] H. Zhou , L. M. Ni, and M. W. Mutka, "Prophet address allocation for large scale MANETs," in *INFOCOM 2003, Twenty-Second Annual Joint Conference of the IEEE Computer and Communications*, Michigan State Univ., USA, Apr. 2003, pp. 1304-1311.

[60] H. Zhou, "Secure Prophet Address Allocation for Mobile Ad-hoc Networks," in *IFIP International Conference on Network and Parallel Computing, 2008. NPC 2008*, Shanghai, 2008, pp. 60-67.

[61] X. Wang and H. Qian, "A Distributed Address Configuration Scheme for a MANET," *Journal of Network and Systems Management, Springer Science+Business Media New York*, vol. 22, no. 4, pp. 559-582, Oct. 2014.

[62] K. Weniger and K. Mase. (2006, Dec.) PDAD-OLSR: Passive Duplicate Address Detection for OLSR, Internet Engineering Task Force (IETF), Internet-Draft. [Online]. https://tools.ietf.org/html/draft-weniger-autoconf-pdad-olsr-01

[63] K. Mase and C. Adjih. (2006, Aug.) No Overhead Autoconfiguration OLSR, IETF Internet-Draft. [Online]. https://tools.ietf.org/html/draft-mase-manet-autoconf-noaolsr-01

[64] D. Kim, H.-J. Jeong, and S. Oh, "Passive Duplicate Address-Detection Schemes for On-Demand Routing Protocols in Mobile Ad Hoc Networks," *IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY*, vol. 58, no. 7, pp. 3558-3568, Sep. 2009.

[65] C. Perkins, E. Belding-Royer, and S. Das. (2003, Jul.) Ad hoc On-Demand Distance Vector (AODV) Routing, IETF, RFC (3561). [Online]. http://www.rfc-editor.org/rfc/pdfrfc/rfc3561.txt.pdf

[66] D. Johnson, Y. Hu, and D. Maltz. (2007, Feb.) The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4, IETF, RFC (4728). [Online]. http://www.rfc-editor.org/rfc/pdfrfc/rfc4728.txt.pdf

[67] M. Gerla, X. Hong, and G. Pei. (2002, Dec.) Fisheye State Routing Protocol (FSR) for Ad Hoc Networks, Internet Engineering Task Force (IETF), INTERNET-DRAFT. [Online]. https://tools.ietf.org/html/draft-ietf-manet-fsr-03

[68] N. C. Fernandes, M. D. D. Moreira, and O. C. M. B. Duarte, "An Efficient and Robust Addressing Protocol for Node Autoconfiguration in Ad Hoc Networks," *IEEE/ACM TRANSACTIONS ON*

NETWORKING, vol. 21, no. 3, pp. 845-856, Jun. 2013.

[69] Y. Rekhter, B. Moskowitz, D. Karrenberg, and G. J. de Groot. (1996, Feb.) Address Allocation for Private Internets, IEFT, RFC 1918. [Online]. http://www.faqs.org/rfcs/rfc1918.html

[70] P. Leach, M. Mealling, and R. Salz. (2005, Jul.) A Universally Unique IDentifier (UUID) URN Namespace, IETF, RFC 4122. [Online]. http://www.ietf.org/rfc/rfc4122.txt

[71] Y. Tang, "Linux-basiertes Testbed zur Evaluierung von Adressierungsprotokollen in Ad hoc Netzwerken," Fakultät für Informatik und Automatisierung, TU Ilmenau Diplomarbeit 2009-11-16 / 111 / II01 / 2235, 2010.

[72] D. C. Plummer. (1982, Nov.) An Ethernet Address Resolution Protocol, IETF, RFC 826. [Online]. http://tools.ietf.org/html/rfc826

[73] R. Morera, A. McAuley, and L. Wong , "Robust Router Reconfiguration in Large Dynamic Networks," in *Military Communications Conference (MILCOM 2003)*, Boston, MA, USA, 2003, pp. 1343-1347.

[74] N. Bulusu, "Self-Configuring Localization Systems," Ph.D., University of California, USA, Los Angeles, 2002.

[75] A. Yousef, H. Al-Mahdi, M. A. Kalil, and A. Mitschele-Thiel, "LHA: Logical Hierarchical Addressing Protocol for Mobile Ad-hoc Networks," in *in Proc. of the 2nd International ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks (PM2HW2N '07)*, Chania, Crete Island, Greece, October, 2007, pp. 96-99.

[76] A. Yousef, A. Mitschele-Thiel, and H. Al-Mahdi, "Analytical Model of the Address Auto-configuration Protocol LHA in Ad hoc Networks," in *The 6th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP08)*, Graz, Austria, Juli, 2008.

[77] A. Yousef, A. Diab, and A. Mitschele-Thiel, "Performance Evaluation of Stateful Address Auto-Configuration Protocols in Ad hoc Networks," in *The 2nd. IFIP international conference on wireless networking and communications, (IFIP Wireless Days 2009)*, Paris, France, Dezember, 2009, pp. 177-182.

[78] A. Yousef, P. Drieß, and A. Mitschele-Thiel, "Comparative analysis of LHA, MANETconf and PROPHET stateful address auto-configuration protocols in ad hoc networks," in *International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN 2009)*, Tenerife, Canary Islands, Spain, 2009, pp. 161-162.

[79] A. Yousef, S. Rishe, and A. Mitschele-Thie, "RNBB: A Reliable Hybrid Broadcasting Algorithm for Ad-Hoc Networks," in *10th International Conference on Wired/Wireless Internet Communications - WWIC 2012, Springer-Verlag, LNCS 7277* , Santorini, Greece, Juni, 2012.

[80] Network Simulator - ns2. [Online]. http://nsnam.sourceforge.net/wiki/index.php/Main_Page

[81] Network simulator Tools. [Online]. https://networksimulationtools.com/computer-network-simulation-projects/

[82] N. Kim, S. Ahn, and Y. Lee, "AROD: An address autoconfiguration with address reservation and optimistic duplicated address detection for mobile ad hoc networks," *Computer Communications*, vol. 30, no. 8, pp. 1913-1925, Mar. 2007.

[83] R. O. Schmidt, A. Pras, and R. Gomes, "On the Evaluation of Self-addressing Strategies for Ad-Hoc Networks," *Energy-Aware Communications, Lecture Notes in Computer Science (LNCS)*, vol. 6955, pp. 31-42, Sep. 2011.

[84] Y. Sun and E. M. Belding-Royer, "A study of dynamic addressing techniques in mobile ad hoc networks: Research Articles," *Wireless Communications & Mobile Computing - Special Issue: Scalability Issues in Wireless Networks—Architectures, Protocols and Services*, vol. 4, no. 3, pp. 315-329, May 2004.

[85] J. W. Tukey, *Exploratory Data Analysis*, 1st ed. Addison-Wesley, 1977.

[86] J. Li, C. Blake, D. J. S., H. I. Lee, and R. Morris, "Capacity of Ad Hoc Wireless Networks," in *ACM SIGMOBILE*, Rome, Italy, 2001, pp. 61-69.

[87] P. Mohapatra, J. Li, and C. Gui, "Multicasting in Ad Hoc Networks," in *AD HOC NETWORKS, Technologies and Protocols*. USA: Springer, 2005, pp. 91-119.

[88] K. Egevang and P. Francis. (1994, May) The IP Network Address Translator (NAT), IETF, RFC 1631. [Online]. http://www.faqs.org/rfcs/rfc1631.html

[89] T. R. Reshmi and K. Murugan, "Secure and Reliable Autoconfiguration Protocol (SRACP) for MANETs," *Wireless Personal Communications, Springer Science+Business Media New York*, vol. 89, no. 4, pp. 1243-1264, Aug. 2016.

[90] J. Postel. (1980, Jan.) DOD STANDARD INTERNET PROTOCOL, IETF, RFC 760. [Online]. http://www.faqs.org/rfcs/rfc760.html

[91] S. Kirkpatrick, M. Stahl, and M. Recker. (1990, Jul.) INTERNET NUMBERS , IEFT, RFC 2131. [Online]. http://tools.ietf.org/html/rfc1166

[92] S. Deering and R. Hinden. (1998, Dec.) Internet Protocol, Version 6 (IPv6), IETF, RFC 2460. [Online]. http://www.ietf.org/rfc/rfc2460.txt

[93] A. Mitschele-Thiel, *Systems Engineering with SDL*, 1st ed. Chichester. West Sussex. PO19 1UD., England: John Wiley & Sons. Ltd, 2001.

[94] M. W. Murhammer, O. Atakan, S. Bretz, L. R. Pugh, and K. Suzuki, *TCP/IP Tutorial and Technical Overview*, 6th ed. USA: IBM Corporation, 1998.

[95] (1996, Apr.) Dynamic RARP Extensions for Automatic Network Address Acquisition, IETF, RFC 1931. [Online]. http://tools.ietf.org/html/rfc1931

[96] R. Gurwitz and R. Hinden. (1982, Sep.) IP - Local Area Network Addressing Issues, Internet Engineering Note (IEN), 212. [Online]. https://www.rfc-editor.org/ien/ien212.txt

[97] J. Broch, D. Maltz, D. Johnson, Y.-C. Hu, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *Proceedings of the Fourth Annual ACM/IEEE International Conference on MobileComputing and Networking (MobiCom)*, Dallas,Texas, USA, October 1998, pp. 85-97.