Design of an IP address Auto-Configuration Scheme for Wireless Multi-Hop Networks

Murimo Bethel Mutanga

200711454

A dissertation submitted in fulfilment of the requirements for the degree of

Master of Science (Computer Science)

Department of Computer Science, Faculty of Science and Agriculture,
University of Zululand

2008

Declaration

I declare that this dissertation is my own original work, conducted under the supervision of Dr. S. S. Xulu and Prof. M. O. Adigun. It is submitted for the degree of Master of Science in Computer Science in the Faculty of Science and Agriculture at the University of Zululand, KwaDlangezwa. No part of this research has been submitted in the past, or is being submitted, for a degree or examination at any other University. All sources used in the dissertation have been duly acknowledged.

Signature: _____

MUTANGA MB

Dedication

To the Mutanga family.

Acknowledgements

I would like to express my appreciation to Dr S. S. Xulu and Prof M. O. Adigun for providing dedicated supervision throughout my research. Financial support by Telkom, Thrip and Huawei to the Department of Computer Science at the University of Zululand through the CoE programme is gratefully acknowledged.

I would also like to thank my research colleagues in the Department with whom I had the pleasure to discuss, argue and work, etc. This work would have been impossible without their contributions. The following made explicit and valuable input that can be found throughout the dissertation. Their help is gratefully acknowledged:

- Dr Victor Waziri for the motivation and help in interpreting mathematical equations;
- Edgar Jembere for valuable suggestions and constructive criticism;
- Siphamandla Nxumalo for helping with the initial simulations of our work carried out using Java;
- Pragasen Mudali, for introducing me to NS2 simulator and for his constructive criticisms;
- Nyandeni Thulani for being my mentor and encouraging me to work hard. Without his help, this work would not have seen light;
- IDINA-HOUR organizers and attendees for valuable suggestions and criticism;
- Tarirai Chani for the encouragement and useful contributions;
- Divya Pillay and Sakhile Ncanana for the help rendered during the dissertation write-up.

I would also like to thank all my team-mates in Judo, Karate and Aerobics for bearing with me for my long absence from training sessions.

My sincere gratitude also goes to my parents and the rest of my family members.

Finally, I would like to thank God for the opportunity given to me, for providing an intellectually and socially stimulating environment, and for giving me the strength to pull through difficult times.

Table of Contents

List of Figures

List of Tables

Abstract

The importance of wireless ad-hoc networks (eg wireless mesh networks) in community and commercial connectivity cannot be underestimated in view of the benefits associated with such networks. An ad-hoc network must assemble itself from any devices that happen to be nearby, and adapt as devices move in and out of wireless range. High levels of self-organization will minimize the need for manual configuration. In essence, self-organization provides an out-of-the-box functionality such that very little technical expertise is required to setup a network. However, efficiently providing unique IP addresses in ad-hoc networks is still an open research question. The goal of this study, on wireless multi-hop networks, was to develop algorithms for IP address auto-configuration. These algorithms should address the following among other problems: Achieving high levels of address uniqueness without compromising on latency and communication overhead.

To achieve the overall goal of this research we proposed changes to the traditional DAD procedure, the Wise-DAD protocol was proposed. We introduced state information maintenance, which is passively collected and synchronized. Passively collecting state information reduced the number of DAD trials thereby reducing latency and communication overhead. Simulations were done in NS-2 to test the performance of the proposed protocol. A comparative analysis was then conducted. Wise-DAD was compared with the Strong-DAD protocol. Experiments on the effect of network size, node density and node arrival rate on communication overhead, address uniqueness and latency were conducted.

Results from the simulation experiments show that Wise-DAD outperforms Strong-DAD in all the three metrics used for performance evaluation. First, Wise-DAD showed better scalability since it performed better than Strong-DAD when network size was increased. Communication overhead in Wise-DAD was generally low whilst the latency was generally uniform. The number of IP address duplicates recorded was reasonably low. Second, Wise-DAD was not affected by node arrival rate on all the three metrics that were recorded. On the other hand, the number of address duplicates in Strong-DAD decreased as the node arrival rate was increased. Interference significantly affected communication overhead recorded in Strong-DAD. Wise-DAD, on the other hand, was not affected by interference. The number of address conflicts in both protocols showed an inverse relationship to interference. However, the number of conflicts for both protocols was significantly different; Wise-DAD recorded much less address conflicts than Strong-DAD.

CHAPTER ONE

INTRODUCTION AND BACKGROUND

1.1 Preamble

In an ad-hoc network nodes collaborate to allow communication without the presence of network infrastructure. Its dynamically changing members and topology require specialized routing. Lack of manual management in ad-hoc networks means that automatic configuration is highly desirable. Automatic configuration of nodes in wireless ad-hoc network will help in reducing administration efforts by users and network administrators. Williams (2002) classifies the requirements for automatic configuration in mobile ad-hoc networking under the following categories: IP interface configuration, translation between host name and IP address, IP multicast address allocation and automatic service discovery. The focus of our research is on IP interface configuration.

In recent years, a lot of research in ad-hoc networks has concentrated on routing protocols. Unfortunately, the same intensity has not been applied to other important related areas, such as node addressing. Routing protocols typically rely on nodes having a unique address (Cavalli and Orset, 2005). In general, nodes often are assumed to have addresses configured a priori, but in ad-hoc networks this is not the case and is not easily accomplished. Nodes require a unique address for packets to be delivered to the correct destination.

Figure 1.1: Schooling fish

"….but birds and fish had no leaders.
Their groups weren't organized that way.
Nobody was directing it.
Nor were individual birds genetically programmed for flocking behaviour…"
(Crichton, 2002)

1.2. Background

The phenomenon of self organization that has its roots in biological and eco-systems is being observed in many other areas. For example, an observation of the behaviour of swarming bees or schooling fish [Figure 1.1] calls for some very interesting conclusions. While the group as a whole exhibits some coherent global behaviour, the individual members that form the group are governed by very simple controls. For instance, the control *"follow in the direction of your neighbours, but do not bump into them"* could be enough for coherent schooling in fish. Other examples of self-organized behaviour can be found in economics, population dynamics, psychology, brain theory, etc. The mechanism, by which the global emergent behaviour relates to the simple, limited, unplanned and unreliable individual agent activities, is quite compelling to computer scientists and the research society as a whole.

A typical example of the emergence of self-organized functions is in the field of ad-hoc networks. Wireless ad-hoc networks are formed by the association of wireless and mobile devices capable of communicating among themselves without any networking infrastructure and dedicated control entity. The automatic configuration

of nodes as they join the network is one of the most important issues regarding the management of wireless ad-hoc networks. In recent years there has been a drive towards self-configuration in ad-hoc networks. This trend will be further accelerated by the advent of ubiquitous computing, where wireless technologies interconnect an increasing number of diverse devices. This will lead to increased complexity and might become a stumbling block for further development. It is clear that a higher level of self-organization will help us to master these challenges (Prehofer and Bettstetter, 2005). High levels of self-organization will help to reduce administrative efforts of users.

The Internet Engineering Task Force (IETF) classified the requirements for zero or auto-configuration in mobile ad-hoc networking under the following categories (Williams, 2002):

a). IP interface configuration. IP interface configuration always includes the configuration of an IP address and netmask; it may include some routing information (such as default route). IP interface configuration is needed before almost any IP communication can take place.

b). Translation between host name and IP address. A zero-configuration name resolution protocol allows users to refer to their devices by name rather than IP address.

c). IP multicast address allocation. IP Multicast is used to conserve bandwidth for multi-receiver bulk delivery applications, such as news.

d). Service discovery. Service discovery protocols allow users or software to discover and select among available services based on service characteristics.

An IP address is one of the most important network parameters in IP networks. Nodes cannot take part in any type of communication if they do not have valid IP

addresses. IP interface configuration is required before other configurations are done since these configurations require that node communicate with each other. We, therefore, focus our attention on IP interface configuration.

Traditionally (in wired networks), an administrator configures each computer manually with an IP address from a finite address space. This approach has no elements of self-organization; instead, it requires significant human intervention and creates a very stiff address structure. A first step toward self-organization was made with the introduction of the Dynamic Host Configuration Protocol (DHCP) (Droms, 1997). Using this protocol, computers are able to automatically obtain an IP address from a server. Although DHCP relieves users of configuring their IP settings, it still requires network administrators to install dedicated DHCP servers (Prehofer and Bettstetter, 2005).

Like in wired networks, nodes in ad-hoc networks cannot take part in any type of communication unless they are configured with an IP address. Although routing protocols assume the existence of unique node addresses, the question of how to provide them remains open. The autonomous nature of ad-hoc networks requires the presence of an IP address auto-configuration mechanism. In these networks, such a mechanism has to cope with a highly dynamic environment and uncertain network structures (Fan et al, 2005). A good IP address auto-configuration protocol should configure a node with an IP address whilst addressing the following goals (Mutanga et al, 2008):

i. ***Low latency for address assignment***
ii. ***Low communication overhead on Address Assignment***
iii. ***Low levels of IP address conflicts***

These goals contradict one another e.g. an improvement towards address uniqueness is usually associated with high communication overhead and high latency.

Many IP address auto-configuration protocols which may be classified under umbrella categories of *stateless* and *stateful* approaches have been proposed in literature. These approaches fail to address all the three goals at once. For example, the main challenge of protocols utilizing a *stateful approach* with a distributed address allocation table e.g. ManetConf (Nesargi and Prakash, 2002), is reliable state synchronization given the uncertain environments that characterize ad-hoc networks (Weniger and Zitterbart, 2004). If inconsistent states exist, address conflicts may occur; address space may get exhausted due to address leaks, etc. However, the cost of explicit coordination is usually higher than the benefits. Explicit coordination of state information is a big challenge and is normally coupled with high communication overhead although in most cases it helps in achieving the desired global property of unique IP addresses. In *stateless approaches,* eg AIPAC (Fazio et al, 2005), AROD (Kim et al, 2007), the most important aspect is the design of the Duplicate Address Detection (DAD) procedures. Stateless approaches degrade dismally when the network grows since the address space shrinks as the network grows. However, according to (Kim et al, 2007) performing a DAD procedure is the best way of making sure that an IP address is unique.

1.3. Statement of the Problem

Simultaneously achieving low latency, a high probability of IP address uniqueness, and low communication overhead is difficult since these goals contradict one another. For example, the best way of making sure that the allocated IP address is unique is to perform a DAD procedure, but on the other hand, the best way to avoid high communication overhead is to eliminate or avoid performing a DAD procedure. The ideal situation is getting maximum benefits (desirable characteristics) while keeping the costs (undesirable properties) associated with attaining those conditions as low as possible but high returns are usually associated with high costs. Therefore, the design must consciously make tradeoffs between these contradictory factors. The task of choosing the best parameters and organizing them to achieve the goal of self-configuration is an optimization problem. Therefore, the questions that this research attempted to answer are:

i. How can we achieve high levels of address uniqueness without compromising on latency and communication overhead?

ii. Which IP address auto-configuration approach or paradigm can yield an optimal protocol performance in terms of low latency, low communication and minimal address conflicts?

1.4. Rationale of the Research

The importance of wireless multi-hop networks (eg wireless mesh networks) in community and commercial connectivity cannot be underestimated in view of the benefits associated with such networks. The growing deployment rate of wireless LANs indicates that wireless networking is rapidly becoming a prevalent form of communication (Sun et al, 2002). High levels of self-organization will minimize the need for manual configuration. In essence, self-organization provides an out-of-the-

box functionality such that very little technical expertise is required to setup a network. This is of paramount importance in rural areas where most people do not have the technical knowledge. Clearly we can not expect people in rural areas, most of whom are illiterate, to configure new nodes. Even if the technical expertise could be found, manually configuring potentially hundreds of devices would be too time-consuming and error prone.

This research has a notable significance if wireless mesh networks are to be adopted as a means of connectivity in both business and communities where uses of the connection vary from user to user or organization to organization. The need for automatic configuration capabilities becomes even more acute when one considers the networked home of the future, with IP-enabled appliances, such as microwave ovens, thermostats, alarm clocks, speakers and various kinds of sensors. In order to allow wide deployment of ad-hoc networks, in which IP routing is the most candidate approach, IP configuration of nodes is a strong requirement that needs to be satisfied (Bernados and Calderon, 2005).

1.5. Research Goal and Objectives

The goal of this study was to take a step in the direction of automatic configuration in wireless multi-hop networks by developing algorithms for IP address auto-configuration. These algorithms should simultaneously achieve low latency, low communication overhead and low address conflicts.

To achieve our overall goal and provide answers to our research questions, we set the following objectives:

i. To investigate the existing IP addressing protocols paying attention at their effect on communication overhead, latency and address uniqueness.

ii. To develop an IP address auto-configuration scheme.

iii. To evaluate the performance of the proposed scheme against existing schemes and show improvements achieved through simulation.

1.6. Methodology

- ***Theoretical analysis of related works by other scholars.***

This part of the research was based mainly on critical evaluative and comparative analysis of existing related works by other scholars. A framework for analyzing related work was formulated. This framework aided in the analysis of and in the development and identification of the appropriate design-paradigm that achieves optimal protocol performance.

- ***Model Design***

Applying self-organization to communications networks requires a constructive engineering approach hence the above analysis was used to formulate the processes, methods or algorithms that contributed towards the development of our model.

- ***Simulation of proposed model using NS2 simulator.***

A design criterion was formulated before the proposed model was simulated in NS2.

- ***Analysis of simulation results.***

A performance analysis of the proposed model was done using both graphical and theoretical techniques. The following metrics were used to evaluate the performance of the proposed model:

- *Latency*

  This refers to the average time taken for a node to be assigned a unique IP address. The address assignment process must be done in as minimum time as possible.

- *Communication Overhead*

  The average number of address assignment packets generated by each node during the address assignment procedure. A good scheme should use as few messages as possible and the communication should be preferably local. Flooding should always be avoided.

- *Address conflicts or Address duplicates*

  The number of duplicated IP addresses in the network. A good scheme should minimize the probability of having more than one node using the same IP address.

Various simulation parameters such as number of nodes, node arrival rate and number of available addresses were varied in order to gain a comprehensive analysis.

1.7 Dissertation Organization

The rest of the dissertation is organized as follows: Chapter two gives a background of this study by introducing some basic concepts in the field of IP addressing. In Chapter three we make an analysis of what others scholars have proposed in the relevant literature. A framework to analyze literature sources is also formulated in this chapter. The chapter concludes by recommending solutions to problems identified in literature. Chapter four provides an architectural design of the proposed model. UML is used to give a clear picture of how the model works. Chapter five details the implementation of the proposed model. It discusses simulation

environment and parameters used in the simulation. An analysis of the results is also given in the same chapter. Chapter six provides a conclusion of the study. It also gives a summary of the achievements made and challenges. The chapter then concludes by suggesting some views and possible future directions.

CHAPTER TWO

TRENDS IN IP ADDRESSING

2.1 Introduction

Internet Protocol (IP) address uniqueness, in IP based networks, is one of the most important requirements since network nodes depend on IP addresses for identification. The Internet Protocol is responsible for routing data packets between networks, and IP addresses specify the locations of the source and destination nodes in the topology of the routing system. For this purpose, some of the bits in an IP address are used for network identification, whilst the other bits identify hosts on the local network. Every data packet transmitted on an IP network contains addressing information in the header. Data packets intended for a single receiver are addressed differently to a data packets intended for multiple receivers. In section 2.2 we give a comprehensive introduction to the field of IP addressing. A brief introduction to the paradigm of automatic configuration in computer networks is given in section 2.3. Automatic configuration of IP addresses in wireless networking environments is discussed in section 2.4. Section 2.5 concludes this chapter.

2.2 IP Addressing

An IP address is a unique identifier used by certain electronic devices in order to identify and communicate with one another on a computer network utilizing the Internet Protocol standard. Any participating network device, (including routers, switches, computers, printers, fax machines, telephones etc) must have its own IP address that is unique within the scope of the specific network. Some IP addresses are intended to be unique within the scope of the global Internet, while others need to be unique only within the scope of an enterprise or local network. IP addresses are assigned to hosts on a network either by specialized software (protocol) or manually by a person. In this section we discuss IP addressing and message transmission in computer networks.

2.2.1 IP Addressing Schemes

The Internet Protocol has two versions currently in use, namely IP version 4 (IPv4) and IP version 6 (IPv6). In this section we give brief analyzes of these two concepts.

(a) IP version 4 (IPv4)

IPv4 uses 32-bit (4 byte) addresses, which limits the address space to 4,294,967,296 (2^{32}) possible unique IP addresses. However, many are reserved for special purposes, such as private networks (approximately 18 million addresses) or multicast addresses (approximately 270 million addresses). This reduces the number of addresses that can be allocated as public Internet addresses. This limitation has helped stimulate the push towards IPv6, which is currently in the early stages of deployment. The format of an IPv4 header is shown in Fig 2.1:

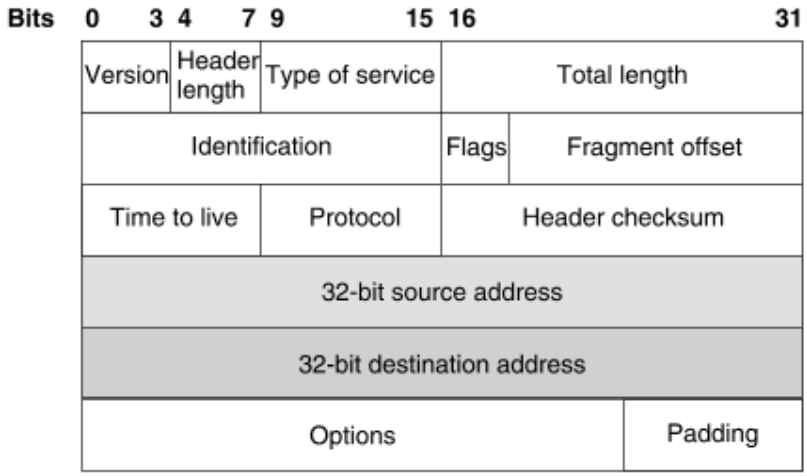

Figure 2.1: IPv4 Header

(b) IP version 6 (IPv6)

IPv6 is designated as the successor of IPv4 for general use on the Internet but it is not yet widely deployed. IPv6 packet headers contain many of the fields found in IPv4 packet headers; some of these fields differ from IPv4. IPv6 addresses are 128 bits (16 bytes) wide, which will more than suffice for the foreseeable future. When IPv6 is fully adopted, there would be exactly 2^{128} unique host interface addresses. Further, this large address space will be sparsely populated, which makes it possible to again encode more routing information into the addresses themselves. The need for such a big address space becomes acute when one considers the networked home of the future, with IP-enabled appliances, such as microwave ovens, thermostats, alarm clocks, speakers and various kinds of sensors. The format of an IPv6 header is shown in Fig 2.2:

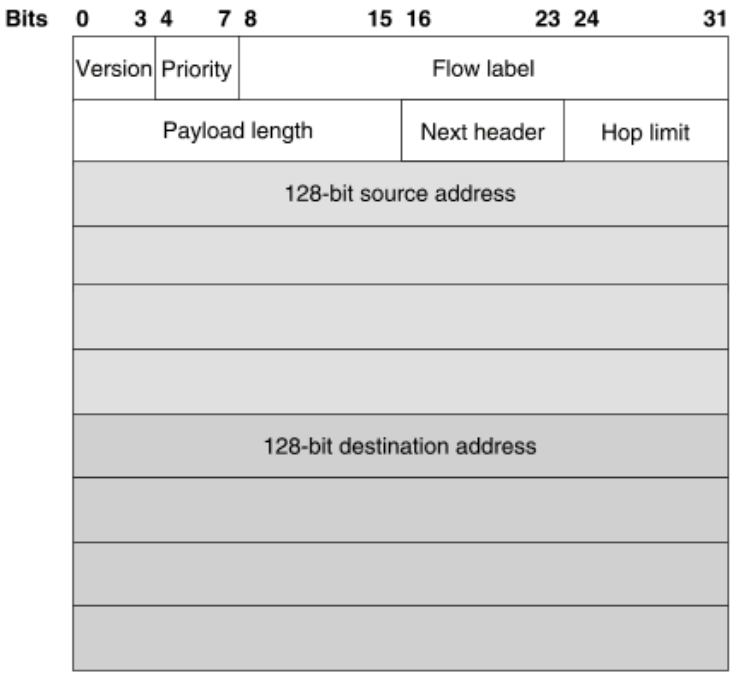

Figure 2.2: IPv6 Header

2.2.2 Static and Dynamic IP addresses

A *Static IP address* is where a computer uses the same IP address every time a user logs on to a network. With a static IP address, a computer's identity can be easily identified by others, and users can easily connect with it. That way, for example, a website, email server, or other type of server connection can be hosted. Static addresses are suitable in planned networks where the number of computers that are going to be part of the network is known. It requires that an administrator manually configures new network nodes with IP addresses. This approach is not suitable in large networks.

In Dynamic IP addressing, an IP address is assigned to a computer, usually by a remote server which is acting as a Dynamic Host Configuration Protocol (DHCP) (Drooms, 1997) server. IP addresses assigned using DHCP may change depending on

the addresses available in the set scope. Although DHCP relieves users of configuring their IP settings, it still requires network administrators to install dedicated DHCP servers (Prehofer and Bettstetter, 2005). This method of IP address configuration can not be directly applied in wireless ad-hoc networks because of the requirement that one node needs to have the capability of configuring other nodes. The first step towards centralized allocation of IP addresses in wireless ad-hoc networks was made with the introduction of the Centralized Auto-configuration (CAC) protocol (Günes and Reibel, 2002). CAC dynamically selects one node that acts as an address-agent (AA) and is responsible for the allocation of IP addresses in the network; hence this node must be permanently reachable.

2.2.3 Message Addressing and Transmission Methods

There are four forms of Message addressing in IP networks:

i. ***Unicast:*** Unicast addressing normally refers to a single-sender-single receiver communication. Usually, a unicast address is associated with a single device or host, but it is not a one-to-one correspondence. Some individual PCs have several distinct unicast addresses, each for its own distinct purpose. Sending the same data to multiple unicast addresses requires the sender to send all the data many times over, once for each recipient. Most applications in the wireless multi-hop networks are based upon unicast communication (Zhou, 2003). Unicast communication requires that nodes have unique IP addresses for packets to be delivered to the correct destination, and due to the routing side effects that may arise from nodes using duplicate addresses (Toner and O'Mahony, 2003).

ii. ***Multicast:*** A multicast address is associated with a group of interested receivers. Addresses 224.0.0.0 to 239.255.255.255 are designated as multicast addresses. This range was formerly called "Class D." The sender sends a single datagram (from the sender's unicast address) to the multicast address, and the routers take care of making copies and sending them to all receivers that have registered their interest in data from that sender.

iii. ***Broadcast*****:** Broadcast is the term used to describe communication where a piece of information is sent from one point to all other points. In this case there is just one sender, but the information is sent to all connected receivers. In the IP protocol, 255.255.255.255 represents a limited local broadcast. In addition, a directed (limited) broadcast can be made by combining the network prefix with a host suffix composed entirely of binary 1s. For example, to send to all addresses within a network with the prefix 192.0.2, the directed broadcast IP address is 192.0.2.255. This type of messaging is used to send control messages in ad-hoc networks hence it doesn't require IP addresses to be unique.

iv. ***Anycast:*** Anycast is a one-to-many routing mechanism. However, the data stream is not transmitted to all receivers (as in broadcast); just the one which the router decides is the closest in the network. Anycast is useful for balancing data loads.

2.3 Automatic configuration in IP networks

Automatic configuration of hosts includes the configuration of IP addresses, subnet masks, default gateway, and other network parameters. Specialized software or protocols are used to configure network nodes. The idea is to provide an out-of-the-

box functionality. Ideally, auto-configuring devices should be "plug and play" such that very little technical expertise is required to setup a network. In this section we discuss the concept of automatic or self configuration in both wired and wireless networks.

2.3.1 Auto configuration in Wired Networks

Auto-configuration in wired networks is achieved by the use of specialized software or protocols such as the Dynamic Host Configuration Protocol (DHCP) (Droms, 1997). The DHCP automates the assignment of IP addresses, subnet masks, default gateway, and other IP parameters. When a DHCP-configured device or computer boots up or regains connectivity after a network outage, its DHCP client sends a query requesting necessary information from a DHCP server. The DHCP server manages a pool of IP addresses and also has information about client configuration parameters such as the default gateway, the domain name, other servers, etc. The query is typically initiated immediately after booting up and must be completed before the client can initiate IP based communication with other hosts. The DHCP server replies to the client with an IP address, subnet mask, default gateway, and other requested information such as DNS server. Depending on implementation, the DHCP server has three methods of allocating IP-addresses:

i. **Manual allocation**, where the DHCP server performs the allocation based on a table with Medium Access Control (MAC) address - IP address pairs manually filled by the server administrator. Only requesting clients with a MAC address listed in this table get the IP address according to the table. This is normally done for security reasons. Only listed MAC addresses are allowed to join the network.

ii. **Automatic allocation**, where the DHCP server permanently assigns to a requesting client a free IP-address from a range given by the administrator.

iii. **Dynamic allocation**, the only method which provides dynamic re-use of IP addresses. A network administrator assigns a range of IP addresses to DHCP, and each client computer on the LAN has its TCP/IP software configured to request an IP address from the DHCP server when that client computer's network interface card starts up. The request-and-grant process uses a lease concept with a controllable time period. This eases the network installation procedure on the client computer side considerably.

2.3.2 Auto configuration in Wireless Multi-hop Networks

Automatic or self configuration in wireless multi-hop networks will help to reduce administrative efforts of users. In wireless multi-hop networks, the requirements for auto –configuration of IP address are classified in under the following headings (Williams, 2002): IP interface configuration, Translation between host name and IP address, IP multicast allocation and Service discovery

i. IP interface configuration

IP interface configuration always includes the configuration of an IP address and netmask; it may include some routing information such as default route. IP interface configuration is needed before almost any IP communication can take place.

ii. Translation between host name and IP address

A zero-configuration name resolution protocol allows users to refer to their devices by name rather than by IP address. Host names are more user friendly than IP addresses and host names have a greater chance of remaining unchanged over time

iii. IP multicast allocation

IP multicast is used to conserve bandwidth for multi-receiver bulk delivery applications, such as audio, video, or news. IP multicast is also used to perform a logical addressing function. For example, when a host needs to communicate with local routers, it can send packets to the all-routers multicast address without having to know in advance the IP address (es) of the router (s).

iv. Service discovery

A service discovery software or protocol must allow users or other software to discover and select among available services based on their characteristics (Gao et al, 2006). This removes the requirement that the user or client software know a server's location in advance in order for the client to communicate with the server. Service characteristics provide a finer granularity of description to differentiate services beyond just the services type. For example, if the service type is printer, the characteristics may be colour, pages printed per second, location, etc.

2.4 IP Address Auto-Configuration in Wireless Multi-hop Networks

The requirement for automatic *IP interface configuration* brings about the problem of IP address auto-configuration. Before any communication can take place, hosts need to be configured with IP addresses. Configuring hosts in ad-hoc networking environments is a complex task due to the uncertainty in these networks. A protocol for assigning IP addresses in wireless multi-hop networks should meet the following requirements:

i. There should be no conflict in IP address assignment, i.e., at any given instant of time there should not be two or more nodes with the same IP address.

ii. An IP address is assigned only for the duration the node stays in the network. When the node departs the network, its IP address should become available for assignment to other nodes after a certain time interval.

iii. A node should be denied an IP address only when the whole network has run out of its available IP addresses.

iv. The protocol should handle network partitioning and merging. When two different partitions merge, there is a possibility that two or more nodes have the same IP address. Such duplicate addresses should be detected and resolved.

v. The assignment of IP addresses to new nodes must be as fast as possible and should avoid flooding the network with a lot of traffic.

2.5 Summary

In this chapter we introduced and explained basic concepts in IP addressing. The concept of message addressing was also introduced. Nodes in IP based networks require IP addresses in order to communicate. Different methods for the allocation of IP addresses in both wired and wireless networks have been proposed in literature. Some of these methods, for example, the Dynamic Host Configuration Protocol (DHCP), are currently in use in today's networks. In ad-hoc networks, however, nodes must be configured automatically without a dedicated entity for this purpose. Automatically configuring nodes in an ad-hoc network is still a challenge. In the next chapter we present and analyze different approaches for IP address auto-configuration in wireless multi-hop networks.

CHAPTER THREE

LITERATURE REVIEW

3.1 Introduction

Most research work in the area of wireless multi-hop networks focus on developing efficient routing protocols. Although routing protocols assume the existence of unique node addresses, the question of how to provide these unique addresses remains open. In order to realize truly spontaneous and infrastructureless networks, a protocol for dynamic allocation of unique addresses is needed (Weniger and Zitterbart, 2004). However, there are still many challenges which must be overcome in order to provide a solution to this problem. In the first chapter we identified the need to achieve high levels of address uniqueness without compromising on latency and communication overhead. Reducing communication overhead and improving on address uniqueness will aid higher level protocols to function well thereby improving QoS provisioning.

In this chapter we discuss a number of IP address automatic configuration schemes that have been proposed in literature. In section 3.2 we analyze the performance of different schemes using a framework in table 3.1. Section 3.2.1 analyzes stateless protocols whilst section 3.2.2 gives an analysis and stateful approaches. In section 3.2.3 we present an analysis of hybrid protocols. Section 3.3 summarizes the limitations of discussed works and concludes this chapter by suggesting ways of overcoming the identified limitations.

Table 3.1: Framework for analysing related work

Communication overhead	The average number of address assignment packets generated by each node during the address assignment procedure. A good scheme should use as few messages as possible and the communication should be preferably local. Some solutions require periodic broadcast of state information thereby resulting in higher communication overhead.
Latency	This refers to the length of the address assignment process. The shorter the process the better. Broadcast normally results in high latency, while local communication results in shorter latency.
Address uniqueness	A unique IP address for all nodes should be an uncompromised requirement. However, guaranteeing uniqueness carries with high costs. We analyze how far different schemes go in trying to achieve this goal. The relationship between address uniqueness and the above two factors (latency and communication overhead) will be the main focus of this research.

3.2. Analysis of Related Work

To critically analyze works by other scholars, we have devised a framework to analyze the performance of different schemes. The framework, given in table 3.1, is based on the goal of this research work i.e. reducing latency and communication overhead without compromising on address uniqueness.

Address assignment paradigms or approaches for ad-hoc networks can be classified into two distinct categories namely stateless and stateful paradigms. Some schemes with characteristics of both approaches are also being developed under the umbrella term of hybrid approaches. Stateless paradigm is also referred to as conflict detection paradigm due to the fact that protocols that follow this approach use a trial and error method to get an IP address. On the other hand, the stateful paradigm is also known as conflict free paradigm since nodes that take part in the address assignment procedure assign addresses that are known to be free. In subsections below, we use the above framework to discuss related work. A critical analysis of the relationship between the above mentioned factors is made in order to gain a comprehensive understanding of the problem.

3.2.1. Stateless Approaches

This method is based on picking an IP address from a pool of available addresses, configuring it as tentative address and verifying the uniqueness of the chosen address. The uniqueness of the address is verified by a Duplicate Address Detection (DAD) procedure, hence most of the research in this approach is aimed at coming up with the best DAD procedure. In case of an address conflict, the node picks a new address and repeats the procedure. In the following subsections we review different stateless protocols.

3.2.1 (a) Strong-DAD (Perkins et al, 2001)

In Strong-DAD, a node simply selects an IP address to use as its address, and checks whether or not it is used in a network using a DAD procedure. If the address is currently in use, the process is started again until a free IP address is obtained. An address is assumed to be free if the timer for a DAD trial expires before receiving a conflict notification message.

Table 3.2: Analysis of Strong-DAD Protocol

Communication overhead	Due to broadcast, Strong-DAD has high communication overhead. It performs a DAD procedure every time a new node requests for an IP address. The number of failed DAD procedures increases as network size increases in size. Due to the increase in the probability of failed DAD as the number of nodes increases, scalability is a problem in Strong-DAD.
Latency	Since the approach uses a time based DAD, address allocation latency depends on the DAD timeout and the number of nodes. If DAD is successful on the first attempt, address allocation latency is equal to the DAD timeout. As the network size increase, latency and communication overhead also increase.
Uniqueness guarantee	This scheme does not specify how it handles the situation of more than one node requesting for the same IP address at the same time hence uniqueness in this scheme is not guaranteed. However, the DAD proposed is likely to get a unique address if all the network nodes are reachable. Strong-DAD does not provide a way for solving the problem of two nodes using the same temporary IP address during the address negotiation process. Also from the birthday paradox, address conflicts are likely to occur when each node chooses its address by random selection (Jeong et al, 2004).

During the IP address negotiation process described above, new nodes use temporary IP addresses for communication. The temporary address is not verified

for uniqueness. As more nodes join the network, communication overhead increases since the number of DAD trials are likely to increase before a free IP address is obtained. An analysis of Strong-DAD is given in table 3.2.

3.2.1. (b) Automatic IP address Configuration (AIPAC) Protocol (Fazio et al, 2005)

The authors proposed a stateless IP address auto-configuration protocol, AIPAC, which is based on Strong-DAD (Perkins et al, 2001). This mechanism avoids the storage of a lot of information about the network in all the nodes, it does not produce too much traffic in the communication channels. Since Strong-DAD does not provide a way for solving the problem of two nodes using the same temporary IP address, AIPAC uses the concept of *Requester* and *Initiator*, which is defined in ManetConf (Nesargi and Prakash, 2002). The Initiator selects an address at random among the allowed addresses, and sends in broadcast a Search_IP packet.

The selected address is specified in the packet. Any node receiving this packet checks whether the address is known (whether this address belongs to it or to another node in its routing tables). If a match is detected, the node sends a Used_IP message to the Initiator. When the Initiator receives the Used_IP message, the address assignment procedure is restarted, and a new address is selected. Conversely, if no reply is received for a given time interval (Search_IP timer), the Initiator sends the Search_IP packet again, in order to face up possible errors in wireless channels. If neither replies arrive, it means that the address is not used yet. The Initiator then notifies the Requester with the NetID of the network and the IP address that it has to use. Table 3.3 gives an analysis of the AIPAC protocol.

Table 3. 3: Analysis of AIPAC Protocol

Communication overhead	Like Strong-DAD, this scheme has high communication overhead. It performs a DAD procedure every time a new node requests for an IP address. The number of failed DAD is likely to increase as the network increases in size. The probability that a DAD procedure succeeds on any given attempt can be calculated as: $$P = \left[1 - \frac{N}{T}\right]$$ Where N is the number of nodes in the network T is the total number of addresses possible. Due to the increase in the probability of failed DAD as the number of nodes increases, scalability is a problem in AIPAC. This affects the overall quality of service of the network and might also increase power consumption of the nodes.
Latency	Since the approach uses Strong-DAD, latency is relatively high. As the network size increase, latency and communication overhead also increase.
Uniqueness guarantee	This scheme does not specify how it handles the situation of more than one node requesting for the same IP address at the same time hence uniqueness in this scheme is not guaranteed. However, like Strong-DAD, the DAD proposed is likely to get a unique address if all the network nodes are reachable.

To detect network merging, AIPAC adopts the idea of a network ID (NetID) used in Prophet (Zhou et al, 2003), and ManetConf (Nesargi, and Prakash, 2002). The authors

then proposed gradual merging when two or more networks with different NetIDs come in contact. Nodes switch from one network to another according to the evolution of the whole system.

3.2.1. (c) Address auto-configuration with address reservation and optimistic DAD (AROD) (Kim et al, 2007)

Kim et al (2007)'s proposal follows the stateless paradigm. The authors argue that it is difficult to guarantee uniqueness of allocated addresses without DAD and went on to propose a distributed auto-configuration scheme that uses address reservation and optimistic DAD. They also argued that reserved addresses help reduce allocation latency and the optimistic DAD guarantees the uniqueness of address with smaller communication overhead. The idea is that configuration time can be reduced by reserved addresses and communication overhead can be minimized by reducing the number of DAD trials. Besides the new and unconfigured node, the authors describe two types of nodes in their scheme:

i. Types 1 node, a node with a reserved address and
ii. Type 2 node, a node without a reserved address.

When both type 1 and type 2 nodes are approached by a new node, they perform DAD but the only difference is that a type 1 node performs DAD after it has allocated a new node with an IP address whilst a type 2 node performs DAD to get a free IP address to give the new node and performs another DAD to look for IP addresses to reserve. A type 2 first tries to borrow an address from a type 1 node and if it fails then it continuously performs DAD until it gets a free IP address. Therefore, a type 1 node performs DAD at most once whilst a type 2 node performs DAD at least once. Figure 3.1 shows how AROD configures a new node and an analysis of this protocol is given in table 3.4.

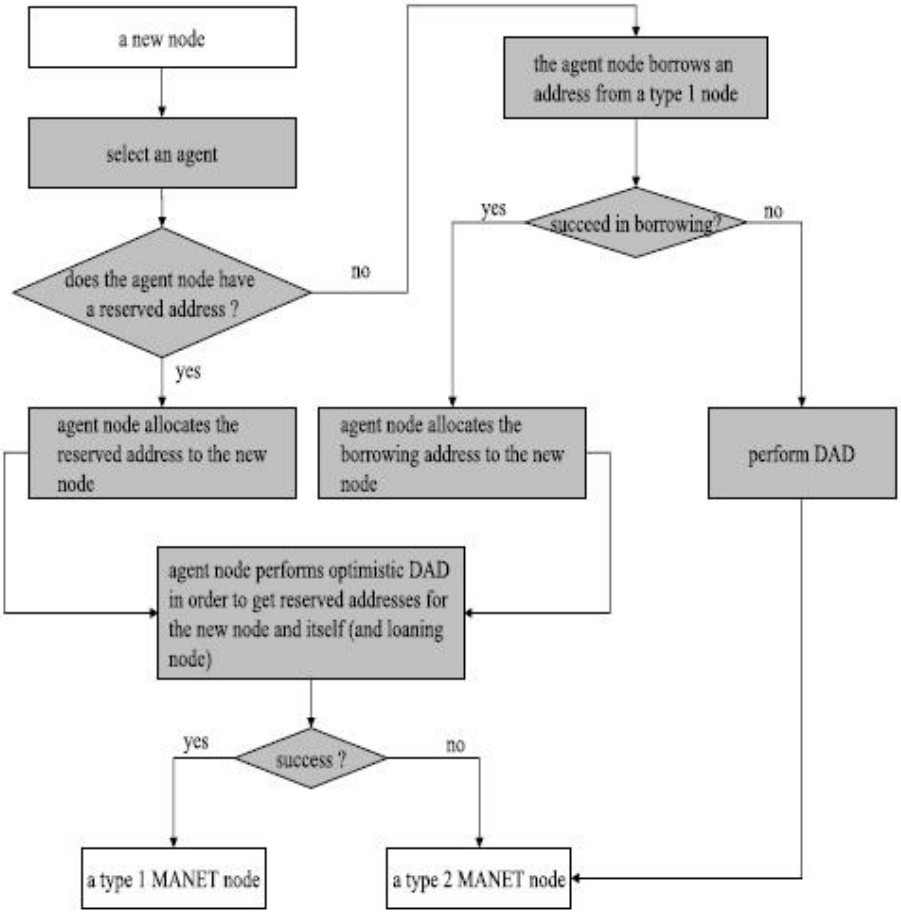

Figure 3.1: AROD allocation process [N. Kim et al, 2007]

Table 3.4: Analysis of AROD Protocol

Communication overhead	Although AROD has a low latency due to the "optimistic DAD" system, it has high overhead since it employs broadcast but the main different with other stateless protocols is that it broadcasts address request messages after it has configured a new node. In a network of say X nodes the number of DAD trials is at least 2X due to the fact that nodes need to have reserved IP addresses.
Latency	Due to the reserved addresses stored at some of the nodes, latency is generally low. A new node is quickly configured. If the Initiator is a type 2 node, DAD follows after the new node has been configured. Even if the initiator does not have a reserved address, it can employ the borrowing mechanism to get an IP address faster.
Uniqueness guarantee	Since no two nodes can reserve the same IP address at any given time, uniqueness is guaranteed. AROD supports simultaneous address assignment especially if the nodes involved (the initiators) in the address assignment process have reserved IP addresses. The problem might arise when the nodes are performing DAD simultaneously.

3.2.1. (d) Weak Duplicate Address Detection (WDAD) (Vaidya, 2002)

The idea behind WDAD is that duplicate addresses may be tolerated as long as packets reach the correct destination, even if the destination node's address is also being used by another node. Each node generates a key at initialization phase, and distributes it with its IP address in all routing messages. Each node maintains keys along with IP addresses in its routing table. When a node receives a routing message with an IP address that exists in its table, it checks if the keys are different. If they are different, a duplicate address is detected and the entry is marked as invalid and additional steps are taken to inform other nodes about this duplication

The main drawback of Weak DAD is its dependency on the routing protocol. It requires some changes to the routing layer to support the introduction of the key. Each node is identified at the routing layer by a kind of virtual address consisting of the IP address and the key value. In addition, WDAD detects address duplication based on local routing information, thus it is totally adapted to proactive routing where each node maintains a complete routing table. For reactive routing, it is not the case; the nodes cache partial routing information for only ongoing and relayed connections which reduces the possibility of detecting address duplication in moderate delays. Table 3.5 overleaf gives an analysis of Weak DAD protocol.

Table 3.5: Analysis of Weak DAD Protocol

Communication overhead	WDAD requires no additional traffic for the auto-configuration mechanism, but the integration of the key value in all data packets increases the bandwidth consumption of every data packet sent in the network.
Latency	Latency is relatively low since address conflicts are tolerated. Nodes are quickly configured on joining the network and address conflicts are dealt with later.
Uniqueness guarantee	Uniqueness is guaranteed by the uniqueness of the key-address combination. This combination is usually unique. The addresses themselves are not guaranteed to be unique. They are tolerated as long as packets reach the destination node intended by the sender, even if the destination node's address is also being used by another node.

3.2.2. Stateful Approaches

There is a central mechanism for allocating IP addresses. In some cases the new nodes will send an IP request packet to its neighbour and that neighbour will choose the IP address for the new node. Auto configuration protocols following this approach can be classified according to the way they maintain the address table. Two approaches used are centralized maintenance of the allocation table, and distributed maintenance of the allocation table.

3.2.2. (a) Prophet (Zhou et al, 2003)

In Prophet, the authors proposed a conflict free (stateful) distributed address configuration scheme that uses a function that produces an integer sequence hence the protocol does not store an address allocation table. The authors argued that IP address auto configuration is the same as assignment of different numbers from an integer range, say R, to different nodes. They went on to argue that if all the addresses that have been allocated and those that are going to be allocated are known in advance, then broadcast could be avoided whilst conflict is still detectable. A way to obtain an integer sequence consisting of numbers in R using a function f(n) was then proposed. The initial state of $f(n)$ is called the *seed*. Different seeds lead to different sequences with the state of $f(n)$ updated at the same time. The basic idea behind Prophet allocation is as follows:

i. The first node, say A, chooses a random number as its IP address and uses a random state value or a default state value as the seed for its $f(n)$;
ii. When another node, say B, approaches A and asks for a free IP address, A uses $f(n)$ to obtain another integer, say $n2$, and a state value, and provides them to B. Node A updates its state accordingly;
iii. Node B uses $n2$ generated by A as its IP address and the state value obtained from node A as the seed for its $f(n)$;
iv. Now node A and node B are both able to assign IP addresses to other nodes.

Address reclamation is not necessary in prophet because the same number will reoccur in the sequence. Nevertheless, the minimal interval between two occurrences of the same number in the sequence is extremely long. The authors say when a node is assigned an old address, X, the previous node with the same address; X is likely to have already left the network. Because of this, this mechanism, does not exclude the

possibility of generating duplicate addresses. The mechanism employed in Prophet works well with short-lived networks like the proposals of Dijkstra et al (2006) and Saxena et al (2005). More analysis of the Prophet protocol is given in table 3.6.

In handling network merging, the authors borrowed idea of partition ID in MANETconf (Nesargi and Prakash, 2002), with a little modification. The first node in the MANET is designated to generate the network ID (NID), which is propagated to new nodes during the course of allocation. Because NID is a random number, if the number of bits for NID is large enough, two networks will have different NIDs. Since some reactive routing protocols (e.g., AODV) require periodic exchange of *HELLO* messages between neighboring nodes, if NID is included in *HELLO* messages, the merger of two separate networks may be easily detected.

Table 3. 6: Analysis of Prophet Protocol

Communication overhead	This scheme does not flood the network with IP request messages. The new node only communicates with its first hop neighbors and IP addresses are generated locally. In this scheme, a new node can get an IP address after sending only one request message. The initiator will use the function to generate an IP address and send a reply. It is clear that the nature of messaging employed by this scheme is highly favourable.
Latency	Since communication is one hop, latency is generally low. This, however, has a negative impact on the scheme's guarantee for address uniqueness.
Address uniqueness	Prophet does not exclude the possibility of generating duplicate addresses because address uniqueness is guaranteed by a sequence of numbers generated by the seed. Thus, prophet may still need the DAD process to avoid address duplicates. The probability of generating duplicate IP addresses increases exponentially when the protocol starts to generate the same sequence again. Because of this, scalability of this protocol is compromised. It is suitable mainly for low scale and short lived networks.

3.2.2. (b) MANETconf (Nesargi and Prakash, 2002)

The authors proposed a system for the management of the IP addresses which is distributed in all the nodes of the network. When a node (requester) is going to enter, it has to rely on a configured node (initiator), which negotiates for an address for it. Each node belonging to the network stores all the used addresses, as well as the ones that are going to be assigned to the new elements. This allows knowing the available addresses at any time. The initiator selects an address among the available ones, and performs a DAD procedure. This is a way for checking whether the same address is being assigned in another part of the network. If all the nodes send a positive reply for this request, the address is assigned.

If a node leaves the network, it has to release its address, by sending a bye message in broadcast. For managing the merging of different networks, a single network ID is used, which is selected by the node with the lowest IP address. When nodes belonging to different networks get in contact, they detect the merging and check for possible duplicated addresses. The system has to verify also if network partitioning occurs. If some nodes do not respond to the subsequent assignment procedure of the IP address, then partitioning is detected. If such nodes also include the one that originally determined the network ID, a new node must be found with the lowest IP address. Since the IP assignment operations may not take place for a long-time, and thus no partitioning can be detected, the node with the lowest IP address must broadcast a message from time to time to show its presence. One cannot easily determine how often this message needs to be sent, since this depends on the dynamics of the network. In table 3.7, we give a detailed analysis of ManetConf protocol.

Table 3.7: Analysis of ManetConf Protocol

Communication overhead	Although MANETconf is a stateful protocol it employs broadcast similar to the one used in stateless approaches. It also requires periodic state information synchronization which is bandwidth consuming. The broadcast mechanism employed is such that in a network with *N* nodes the initiator receives a total of (*N*+1) messages. If there is another initiator trying to allocate the same IP address the initiator is likely to receive (2*N*+1) messages. This scheme, therefore, has poor messaging system.
Latency	The length of the IP address assignment process in MANETConf is proportional to the network size because every node in the network takes part in the address assignment process. Scalability is severely compromised because of this.
Address uniqueness	Uniqueness is guaranteed by first checking whether the same address is being assigned in another part of the network and by keeping state information up-to-date. If inconsistent states exist, the likelihood of duplicate IP addresses is very high.

3.2.2. (c) Buddy protocol (Mohsin and Prakash, 2002)

The Buddy Protocol is based on the concept of binary split. In the beginning, there is only one node in the network that has the entire pool of IP addresses. New nodes are given their blocks of unassigned IP addresses to give out to other nodes so every node has a disjoint set of IP addresses that it can assign to new nodes without consulting any other node in the network. Nodes synchronize from time to time to keep track of the IP addresses assigned and detect any leaks in the available pool of IP addresses. Leaking is mainly caused by nodes that leave the network abruptly whilst holding a pool of free addresses. The synchronization process involves every node broadcasting its pool of IP addresses. This broadcast is used by every other node to update its IP address allocation table.

To address network merging, the concept of partition ID is used. If a network gets partitioned into two, as long as they do not run out of IP addresses, they do not generate new partition IDs, so if they merge later there are no duplicated addresses. If one of the partitions runs out of addresses, it generates a new partition ID and acquires the IP address blocks that belongs to the other partition. If these partitions merge later, one of the partitions has to give up its IP address block. In this mechanism, the one with the larger address block is the one that gives up it addresses and has to acquire new ones. A detailed analysis of the Buddy protocol is given in table 3.8.

Table 3.8: Analysis of Buddy Protocol

Communication overhead	This scheme suffers from high overheads due to the requirement that each node has to broadcast its pool of addresses for state synchronization. Inconsistent states result in address conflicts and an exhausted address space due to address leaks. The frequency of the broadcast is also difficult to determine due to the unpredictable nature of ad-hoc networks. If nodes join and leave the network more rapidly, synchronization must be done faster whilst on the other hand if a network is stable there synchronization might not be necessary.
Latency	Since the address allocation process can be done locally, latency is very low.
Uniqueness guarantee	Address uniqueness depends on the accuracy of state information. If state information is accurate, no two nodes can have the same IP address. The accuracy of the state information on the other hand depends on how often synchronization takes place and the stability of the network in general. However, frequent synchronization leads to high communication overhead.

3.2.3. Hybrid Approaches

Hybrid protocols combine elements of both stateful and stateless approaches.

Protocols that follow this approach combine DAD with either a centrally maintained or a distributed common allocation table. *Hybrid Centralized Query-Based Autoconfiguration* (*HCQA*) protocol (Sun and Belding-Royer, 2003) utilizes Strong-DAD together with a centrally maintained allocation table. The *Passive Autoconfiguration for Mobile Ad-hoc Networks* (*PACMAN*) protocol (Weniger, 2005) combines PDAD (Weniger, 2004) with a distributed maintenance of a common allocation table such as the one proposed in MANETConf (Nesargi and Prakash, 2002).

3.2.3. (a) The HCQA protocol (Sun and Belding-Royer, 2003)

HCQA utilizes Strong-DAD (Perkins et al 2001) mechanism along with a centrally maintained allocation table in order to improve address consistency hence it can be classified as a hybrid protocol. When a node wants to join the network, it chooses two addresses, a temporary and a tentative one, and performs a DAD procedure similar to Strong-DAD. If the address auto-configuration is successful, the new node must register its tentative address with an Address Authority (AA). Therefore, it waits for an advertisement of the AA for a certain period of time. Upon receiving the advertisement from the AA, the new node launches a registration request and waits for the registration confirmation. Only after the confirmation, the node may begin to use this address. After a successful registration, the node runs a timer and reinitiates the registration process each time the timer expires.

In addition to holding the states of all assigned IP addresses, the AA can help in detecting address duplication in the initialization phase by replying to address request destined to a used tentative address. When nodes initialize, the first node that obtains a unique IP address becomes the AA in the network. The AA chooses a unique identifier and broadcasts it periodically to identify the network. If a node does not receive any AA advertisement for a certain period, it considers that there is network partitioning and becomes the new AA and generates a new network identifier. When a node receives a new network ID, it must register its address with the new AA, thus no address change is needed. Network merge is detected by the presence of two network IDs. In this case, only the AAs are involved in detecting address conflicts by exchanging their different tables. To reduce the centralization at the address authority, the protocol specifies a mechanism to backup the address authority's address table. To do so, the AA picks the first node that has registered its address as the AA backup. Every time a new node registers its IP address with the AA, the AA sends an update with the new information to the AA backup. Table 3.9 gives an analysis of the HCQA based on the framework given in table 3.1.

Table 3.9: Analysis of HCQA Protocol

Communication overhead	The number of failed DAD procedures increase as the network size increases. Due to the increase in the probability of failed DAD as the number of nodes increases, scalability is a challenge in HCQA. All nodes are also required to periodically broadcast a message to the AA to reinitiate the registration process each time the timer expires. This mechanism overloads the AA with a lot of traffic. This affects the overall quality of service of the network and might also increase power consumption of the nodes especially the AA.
Latency	Since the approach uses DAD, latency is relatively high especially when the network size is large.
Uniqueness guarantee	This protocol has low address conflicts. Its strength is that it adds robustness to the Strong-DAD mechanism, by keeping and synchronizing state information. Even if a node is temporarily unreachable, the AA can defend its IP address. However, this has a negative effect on the communication overhead and latency.

3.3. Summary

A number of protocols utilizing different approaches have been proposed for the purpose of address auto-configuration in wireless ad-hoc networks. The main challenge of protocols utilizing a stateful approach with a distributed address allocation table is securing reliable state synchronization mechanism. Inconsistent states result in address conflicts, an exhausted address space due to address leaks or unnecessary address changes. Most protocols rely on periodic flooding, which consumes a considerable amount of bandwidth. A centrally maintained allocation table has fewer problems with inconsistent states, but requires a dynamic leader election algorithm. State recovery after the central entity has changed is an inherent problem, especially in highly dynamic scenarios. Overloading the central node is another challenge that is difficult to overcome.

An interesting approach seems to be the prediction of conflicts by generating addresses using a well-known function, for example Prophet (Zhou, 2003), but the existence of a reliable function for realistic scenarios remains an open issue. Prophet cannot work in large networks and where nodes rapidly join and leave the network.

The most important component of stateless approaches is DAD. Due to broadcasting, stateless approaches degrade dismally as the network grows since the address space shrinks as the network size increases. However, according to Kim et al (2007), performing a DAD procedure is the best way of making sure that an IP address is unique. The probability, P, of getting a free IP address on a given DAD procedure can be calculated as:

$$P = \frac{S - N}{S},$$

where S is the address space, ie the number of possible or valid IP addresses in the network, N is the number of nodes in the network. It is clear that communication overhead increases as the network size increases since the number of DAD trials are likely to increase before a free IP address is obtained. This is so because as $N \to S$, P becomes very small.

A passive approach of detecting address conflicts (e.g. Weak DAD) looks promising, because it does not consume any additional bandwidth. However, special algorithms must be developed for every routing protocol since such mechanisms depend on routing protocols for conflict detection. It, therefore, remains unclear if there are algorithms for all routing protocols to enable reliable detection of all conflicts. Hybrid protocols combine elements of both approaches by maintaining an allocation table and performing a DAD, which can increase the robustness but might be too complex for resource constrained devices.

In this work, we support Kim et al, (2007)'s notion that it is difficult to guarantee uniqueness of allocated addresses without DAD, and Campos and Ricardo (2005)'s notion that stateful frameworks are difficult to implement. However, a solution that decreases the chances of DAD failing in large networks is required to reduce latency and communication overhead. In the next chapter we propose changes to the traditional DAD procedure (Strong-DAD) which will reduce both communication overhead and latency whilst not compromising on the uniqueness of the allocated address.

CHAPTER FOUR

MODEL FORMULATION

4.1 Introduction

We have identified the need to achieve high levels of address uniqueness without compromising on latency and communication overhead. Solving this problem will, among other things, help in providing Quality of Service (QoS) guarantees in the network. Reducing communication overhead and improving on address uniqueness will aid higher level protocols to function well, thereby improving QoS provisioning. However, in chapter three we concluded that it is difficult to guarantee uniqueness of allocated addresses without performing a Duplicate Address Detection (DAD) procedure. Unfortunately, schemes with DAD procedures require longer time and more packets to allocate addresses (Weniger and Zitterbart, 2004). We, therefore, propose changes to the traditional DAD procedure which will reduce both communication overhead and latency whilst ensuring the uniqueness of the allocated address.

In our approach, we introduce state information maintenance which is passively collected but not actively maintained. Passively collecting state information will reduce the number of DAD trials thereby reducing latency and communication overhead. Reducing communication overhead helps to conserve bandwidth thereby improving on QoS of the whole network. This chapter presents the ***Wise-DAD*** protocol developed in our work to address problems identified in chapter 3. Section

4.2 outlines the design criterion that was followed in Wise-DAD. A detailed design of this protocol is given in section 4.3. Section 4.4 concludes this chapter.

4.2. Design Criteria for an IP Address Auto-configuration Protocol

From our review of literature we identified the following design criterion to follow when designing IP address auto-configuration protocols: distributed algorithm, network merging, algorithm complexity, address re-usability, low delay, address uniqueness and low communication overhead. In the subsections below, we justify this design.

4.2.1 Distributed Algorithm

The choice of a distributed algorithm alleviates the need for instituting a process for the election of a central node that performs the address allocation process. The responsibility for address configuration has to be borne by all nodes that are already part of the network. There must not be a single Dynamic Host Configuration Protocol (DHCP) server since it is impossible to guarantee that the server will always be available. All nodes should collectively perform the functionality of a DHCP server.

4.2.2 Network Merging

The protocol should handle network partitioning and merging. When two different partitions merge, there is a possibility that two or more nodes have the same IP address. Such duplicate addresses should be detected and resolved.

4.2.3 Computational Complexity

The algorithms used must be simple since some computations are beyond the computational capacity of the low cost, resource-constrained wireless multi-hop network nodes used in low cost connectivity. Artificial intelligence techniques must be avoided since they are computationally complex. Complex mathematical calculations should also be avoided.

4.2.4 Address Re-use

An IP address should be assigned only for the duration the node stays in the network. When the node departs the network, its IP address should become available for reassignment to other nodes. This allows for easy network expansion. A node should be denied an IP address only when the whole network has run out of free IP addresses.

4.2.5 Low Latency

The address assignment process must be done in as minimum time as possible to avoid unnecessary delays.

4.2.6 Low Communication Overhead

Because every successfully received packet, must have consumed bandwidth (and power as well), a good scheme should use as few messages as possible and the communication should preferably be local to avoid overwhelming the network with traffic.

4.2.7 Low Address conflicts

Nodes require a unique address for packets to be delivered to the correct destination, and due to the routing side effects that may arise from nodes using duplicate addresses (Toner and O'Mahony, 2003). A good scheme should minimize the possibility of having more than one node using the same IP address.

4.3 The Wise Duplicate Address Detection (DAD) protocol

In this section we give a description of the Wise-DAD protocol devepoled in this research work. Section 4.3.1 gives a brief outline of the protocol. In section 4.3.2 we present the system architecture of the Wise-DAD protocol.

4.3.1 Brief Description of Wise-DAD

The Wise-DAD protocol works as follows: An unconfigured node periodically broadcasts a *request to join message.* If there is another unconfigured node within its transmission range, a network is automatically formed. The node with the lower Host identifier (HID) chooses network parameters; gives the other node an IP address and other configuration details. The HID is a randomly generated temporary IP address used by nodes before they acquire permanent IP addresses. If a configured node receives a *request to join message*, it assumes that an unconfigured node wants to join the network hence it will offer to act as its initiator by sending a *confirmation message*. The new node then selects only one of its neighbors node to act as its negotiating agent (initiator). It sends a *select initiator message* to the first node to respond. The initiator then generates a random IP address from the allowed addresses and checks its allocation table if there is no node in the network that have requested for or used the same IP. If the address is not known, the initiator then performs a DAD (using an address request message).

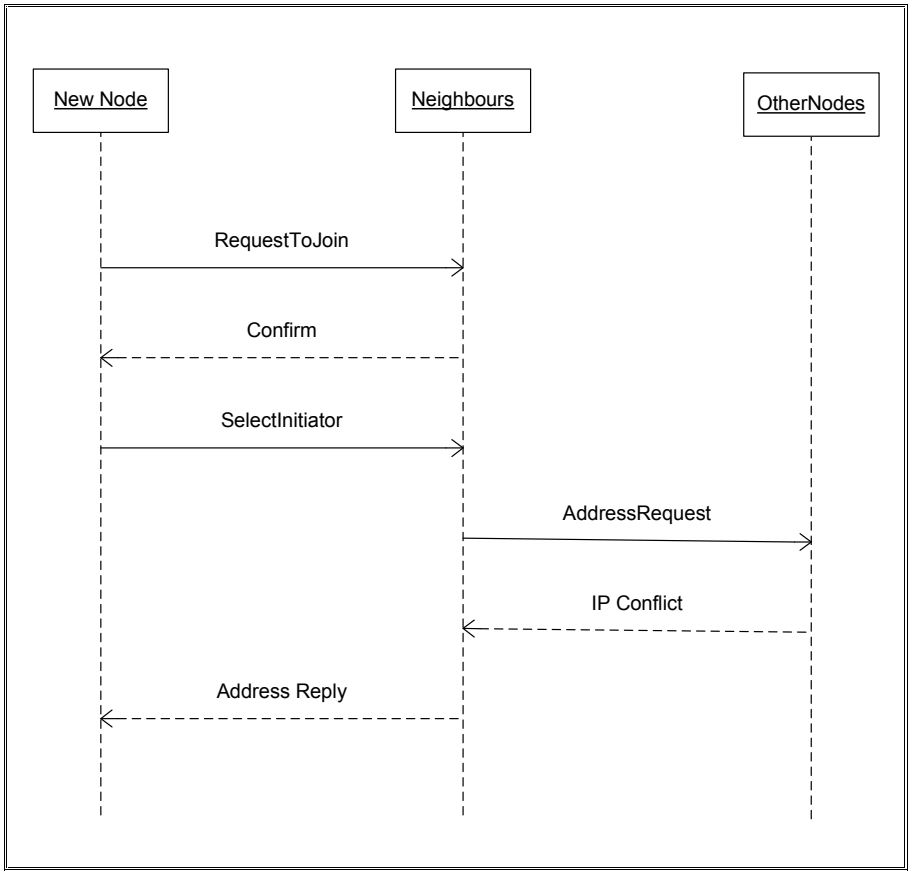

Figure 4.1: Summary of address allocation process in Wise DAD

All nodes receiving an address request packet update their tables and add their IP addresses to the packet before broadcasting it. If any node is using the requested address, it defends it with an *IP conflict message* and this process is repeated. If no *IP conflict message* is received after a certain time interval, the address is assumed to be free and the initiator will send an *address reply message* to the new node. The address reply message will have the IP address for the new node, the network identifier (NetID) and the state information (allocation table). Figure 4.1 above summarizes this process.

If a node leaves the networks gracefully, it broadcasts a goodbye message and all the nodes delete its IP address from their allocation tables. If a node leaves abruptly, immediate address reclamation is not performed. Since the node will not be sending or forwarding any data packets, other nodes will remove all passive nodes from their allocation tables. Allocation tables are not actively synchronized; they are used only as an estimate of the state information. If a node does not take part in an IP address allocation process for a long time, its IP address will be deleted when the size of the allocation table reaches a certain level because it will be assumed that the node left the network abruptly.

Network merging is handled by using the concept of partition or network identifiers proposed in MANETConf (Nesargi and Prakash , 2004). Using this approach each network uses a randomly generated network identifier that is stored by each node. Nodes periodically send one hop broadcast messages of their network identifiers. The network identifier can be incorporated in the hello messages of the routing protocol. If a node receives a hello message with a different network identifier, network merging is detected. All nodes in the network with the lower network identifier relinquish their IP addresses and starts the IP address negotiation process afresh. This mechanism allows for fast network merging to take place.

4.3.2 System Architecture of Wise-DAD

In this section we give a full description of the Wise-DAD protocol. We divided the problem of IP address auto-configuration and maintenance according to the following functions; *(a) Network formation:* In this section we describe how a network is formed. We also explain how the nodes decide who chooses network parameters

such as the network identifier. *(b) Node admission:* We describe how nodes join the network i.e. how new nodes acquire IP addresses. *(c) Node departure:* This function is responsible for determining how addresses for nodes which are no longer part of the network are reclaimed and reused. *(d) Network Merging:* This function is concerned with the process of detecting and handling network merging. Address conflict resolution is also part of this function.

(a) Network formation

A new node broadcasts a *request to join message* and sets a timer (RTJ_Timer). The message is destined only to the immediate neighbours (one hop neighbours). When the timer expires (after 10 seconds), the node will rebroadcast the message and reset the timer again until it receives at least one reply from its neighbouring nodes. If two unconfigured nodes get within each other's transmission range, a network may be started. Initially, an unconfigured node uses a temporary IP address, Host identifier (HID). Upon receiving a request to join message, an unconfigured checks the HID of the sending node. The node with the lowest HID chooses the Network identifier (NID) and sends it to the other node together with an IP address that it chooses at random. The second node will also choose its own IP address and notify the other node of its chosen IP address. From that point, a network of two nodes starts to exist. If one of the configured nodes receives a request to join message, a reply with its availability to act as an initiator is sent. This process is explained in detail in (b) overleaf. Figure 4.2 outlines the steps followed when a node receives a request to join message during the process of network formation. Diagrammatic representation of this process is given in Figure 4.3.

```
function receive : REQUEST_TO_JOIN
Begin
        if (packet_type = REQUEST_TO_JOIN) then
        Begin
                 If (not configured) then
                Begin
                    If (HID < message_HID) then
                    Begin
                            GenerateNID()
                            SendMessage(AddressReply)
                    End
                End
                Else If (configured) then
                    SendMessage(confirmation)
        End
       Else call appropriate handler
    End
```

Figure 4.2: Processing a request to join message

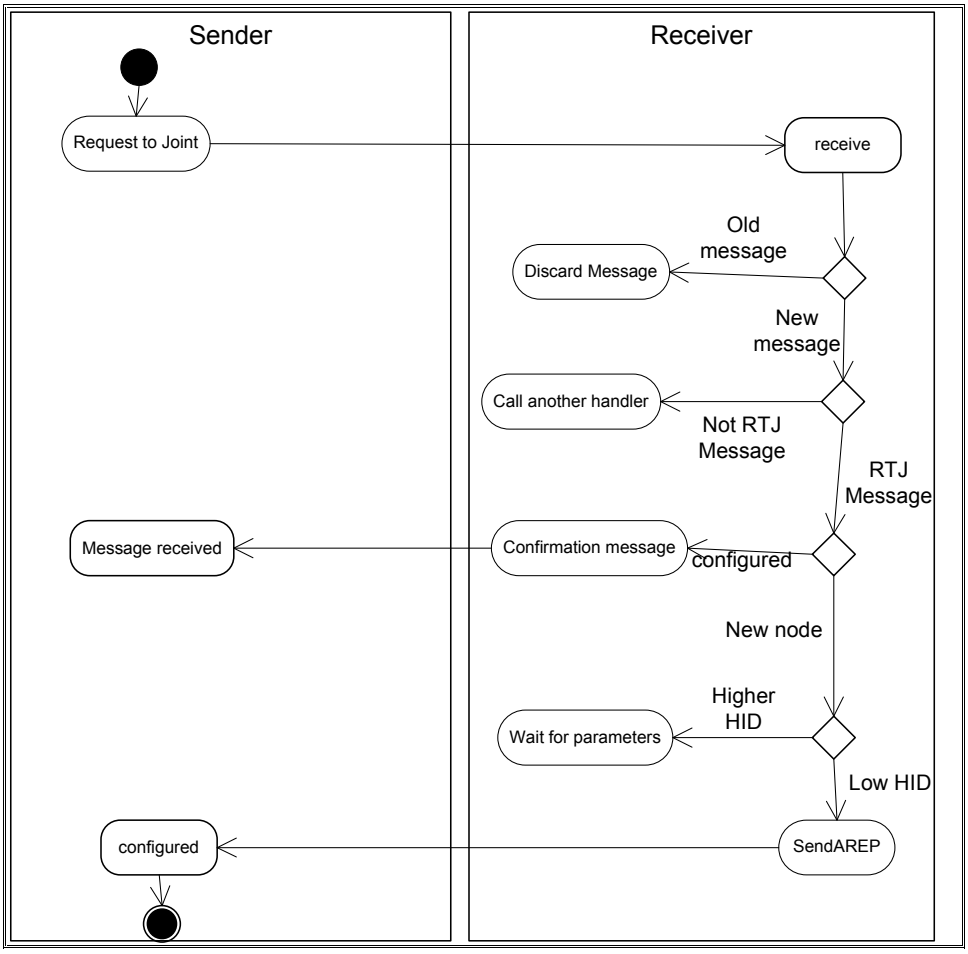

Figure 4.3: Wise DAD Network formation process

(b) Node Admission

A configured node receiving a request to join message replies with a confirmation message to the sender. If a new node receives more than one *confirmation message*, it takes the first one and ignores the rest. The new node then replies with an *initiator-selection message*. The initiator generates a random IP address and checks if it is in its allocation table before it starts the negotiation process. If the address is in the allocation table, it generates another one otherwise it will broadcast an *address request* (AREQ) message and set a broadcast timer. If the broadcast timer expires (after 1.8 seconds) without any node defending the requested IP address, the initiator sends an *address reply* (AREP) message to the new node. On receiving an address request message other network nodes first check if the message is new or not before checking if the requested IP address has been assigned them. A message sequence number is used to determine if a message is new or not. If the address is found to be in use, an IP conflict is sent to the initiator and the process is repeated. If the message is not new, it is discarded, otherwise it will be broadcast.

Before the message is broadcast, the recipient adds it IP address to the message. As the message is passed from one node to another, a reverse path to the initiator will be contained in the packet. This allows for an IP conflict message to be sent back to the initiator. When nodes receive AREQ, they also update their allocation tables using IP addresses in the reverse path list before rebroadcast the AREQ. Figure 4.4 and Figure 4.5 overleaf, respectively outline the steps followed when a node receives a confirmation message and address request (AREQ) respectively. A diagrammatic representation of the address negotiation process is given in Figure 4.6.

```
Function receive : Confirmation
Begin
      If (packet_type = confirmation) then
       Begin
            MessageCount = MessageCount + 1
            If (messageCount > 1) then
              Discard message
            Else
              sendMessage(Confirmation)
        End
      Else call appropriate handler
End
```

Figure 4.4: Processing confirmation message

```
Function receive : AREQ
Begin
     If (packet_type = AREQ) then
     Begin
          If (message is old) then
            Discard message
           Else
                Begin
                     If (requested address  = My_addrress) then
                       SendMessage(IP_Conflict)
                     Else
                          If (requested address != my_address) then
                        Begin
                               Add_my_address_ToPacket
                               BroadcastAREQ
                               updateAllocationTable
                          End
                End
     End
    Else  call appropriate handler
End
```

Figure 4.5: Handling address request message

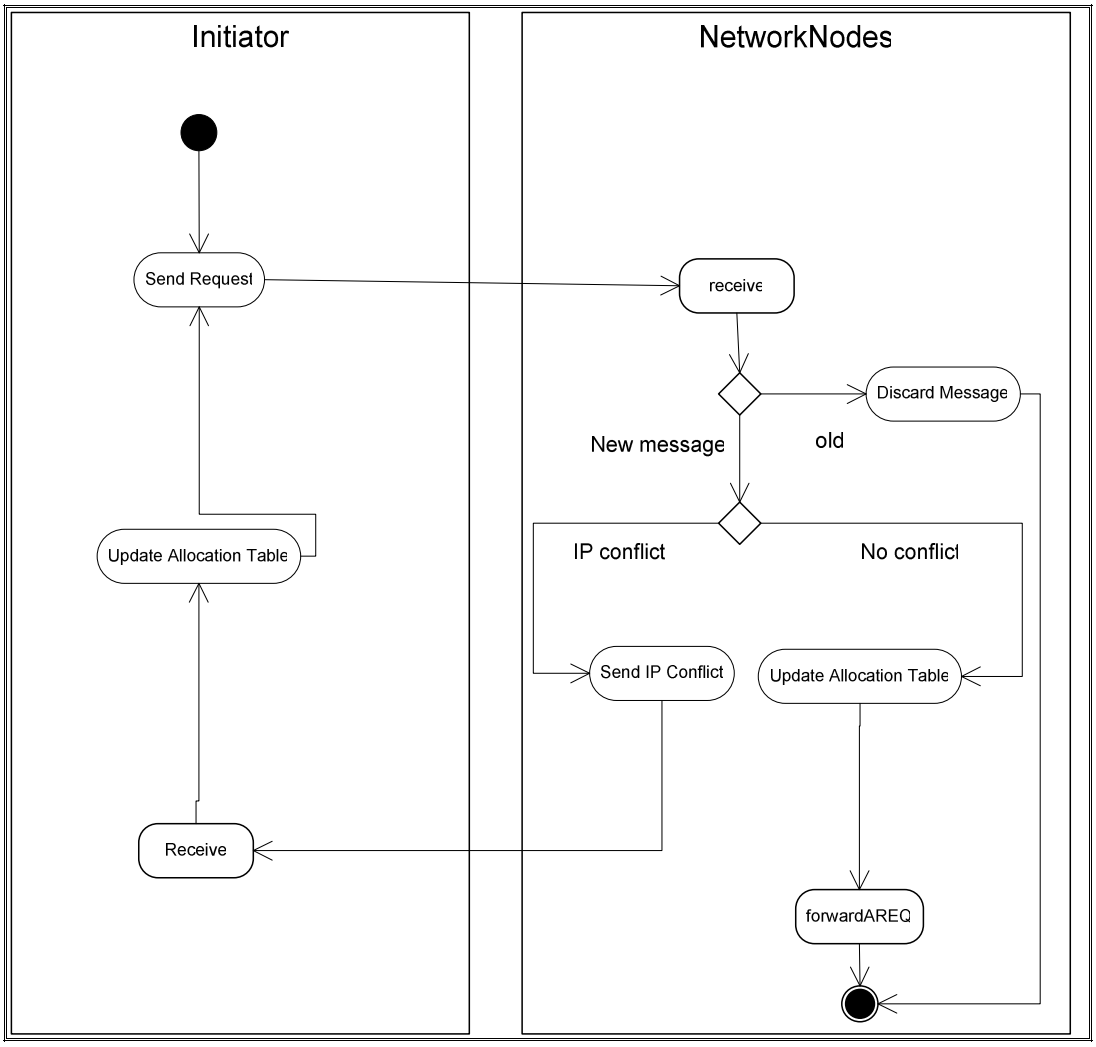

Figure 4.6: Address configuration process

(c) Node departure

If a node departs gracefully, it informs the other network nodes by broadcasting a goodbye message. Nodes receiving a goodbye message delete the departing node's address from their allocation tables. This allows for re-use of the address by new nodes. Before processing this message nodes first check whether the message is new or not. If the goodbye message is not new, it is discarded otherwise it will be broadcast. Before the message is broadcast, the node adds its IP address to the message. When nodes receive a goodbye message, they also update their allocation tables using IP addresses contained within the packet.

```
Function Receive: Goodbye
Begin
    If (packet_type = goodbye) then
    Begin
        If  (goodbye_message is old) then
         Discard message
         Else
            Begin
                If (Departing Node IP_addr in MyAllocationTable)  then
                UpdateAllocationTable
            End
     End
    Else call appropriate handler
End
```

Figure 4.7: Processing of goodbye message

However, node departures can be abrupt. In that case the departing node will not have the time to inform other nodes of its departure. If a node does not take part in any IP address assignment process, it is assumed to have left the network and its IP address is deleted from address allocation tables. If the address allocation table reaches a certain level, all passive nodes are deleted. The deleted IP addresses will be tried in subsequent address assignment procedures. Figure 4.7 explains how nodes handle the goodbye message, whilst Figure 4.8 shows the graceful node departure process.

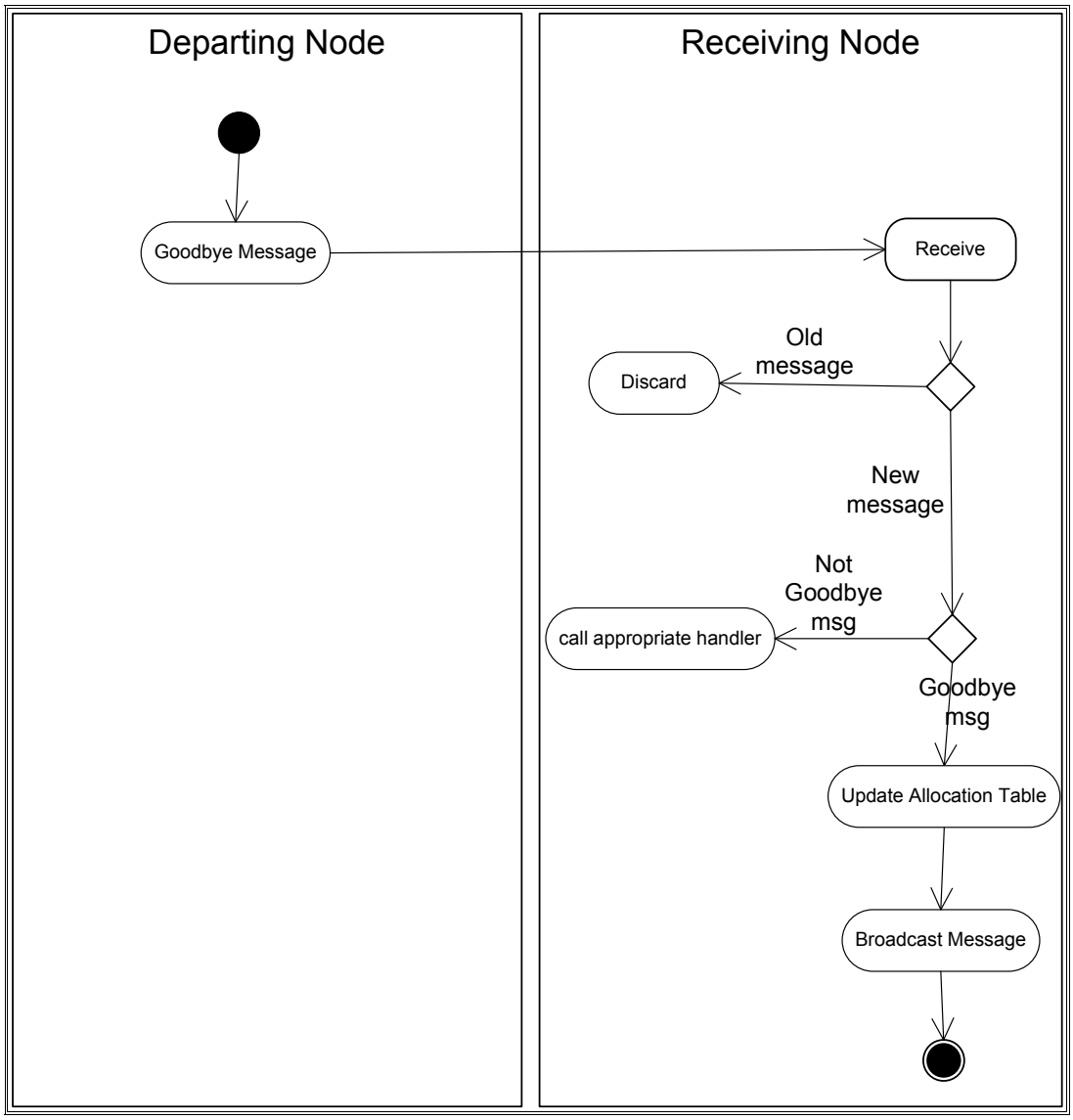

Figure 4.8: Node departure process

(d) Network Merging

In Wise-DAD we adopted the concept of network IDs used in MANETConf (Nesargi and Prakash, 2004) to handle network merging. The first node in the network generates a random network identifier to be used by all subsequent nodes in the network. Network IDs are not used for routing as in wire line networks, but to differentiate between different networks in case of mergers. Routing in wireless multi-hop networks only consider the local IP address (Indrasinghe et al 2006).

```
Function receive : mergerMessage
Begin
    If (packet_type = mergerMessage) then
    Begin
        If (mergerMessage is old) then
        Discard message
        Else
            Begin
                Set NetID = 0
                Set Configured = 0
                ClearAllocationTable
                Broadcast(Request_To_Join)
            End
    End
   Else  Call appropriate handler
End
```

Figure 4.9: Processing network merging message

Nodes periodically broadcast *hello messages* with their network identifiers. If a node receives a *hello message* with a network ID which is lower that its own, it initiates the network merging process by broadcasting a *merger message* to inform the other nodes of the impending merger. All nodes in the network with the lower network ID relinquishes their IP addresses and starts the process of IP address requisition afresh. Figure 4.9 explains how the network merging message is processed in the receive function whilst Figure 4.10 shows the network merging process. The receive function is implemented on all the nodes to handle incoming packets.

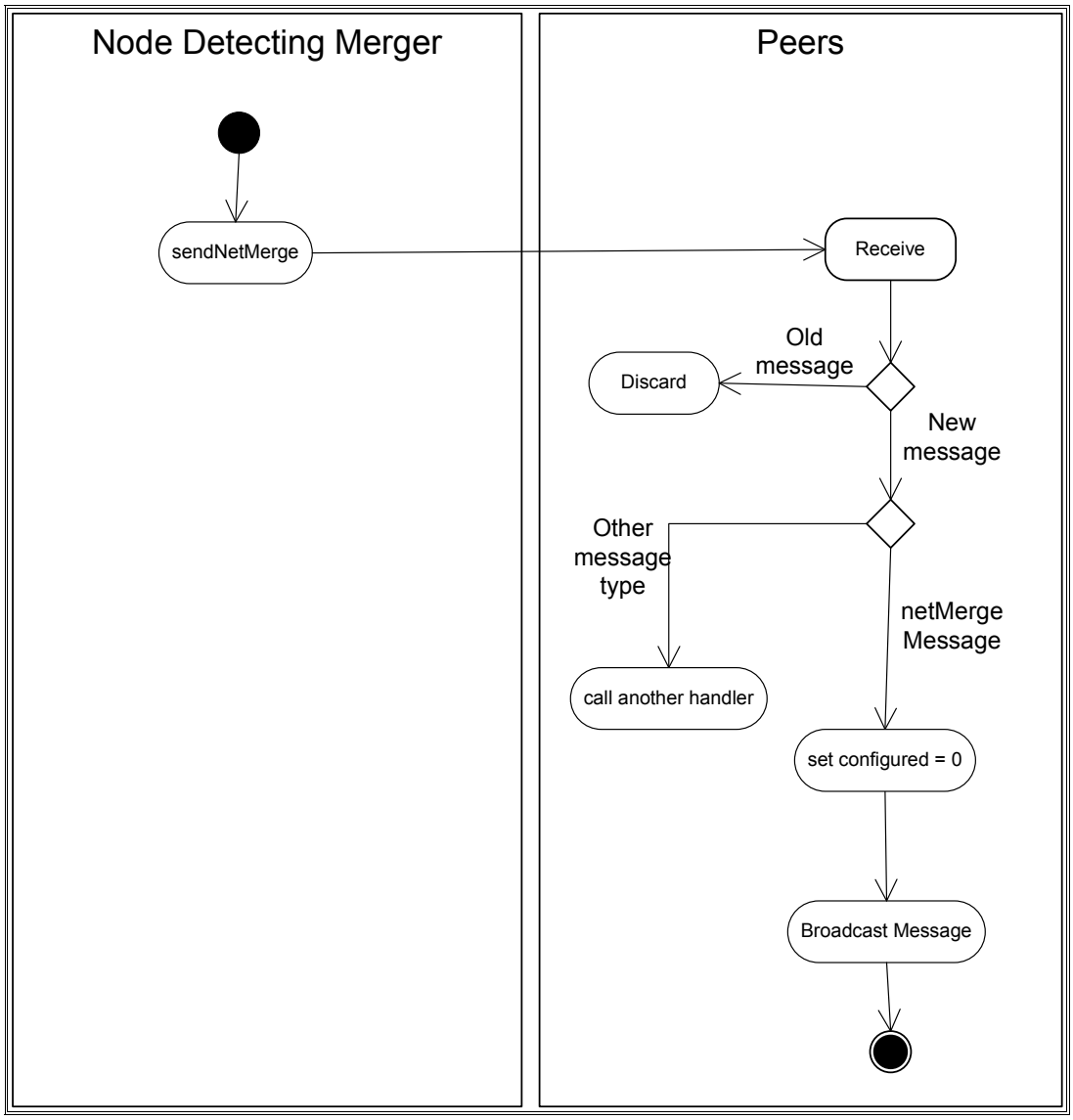

Figure 4.10: Network merging process

4.4 Summary

In this chapter we presented the Wise-DAD protocol for IP address allocation in wireless multi-hop networks. Wise-DAD follows the stateless paradigm but maintains a passively synchronized state to reduce the number of DAD trials. The system architecture of Wise-DAD consists of the following four main components: network formation, node admission, network merging and node departure. We used algorithms and UML diagrams to give a detailed explanation of how the different components work and interact. Simulation experiments and performance evaluation of the proposed protocol are presented in the next chapter.

CHAPTER FIVE

PERFORMANCE EVALUATION OF THE WISE-DAD PROTOCOL

5.1 Introduction

In the previous chapter, we presented the Wise-DAD protocol for IP address auto-configuration. In this chapter we present simulation results of the proposed protocol. Wise-DAD follows the stateless paradigm with soft state maintenance to reduce the number of DAD trials. We considered a stand-alone ad-hoc network, i.e., an ad-hoc network with no connection to an external network like the Internet. It was assumed that the IP address block from which nodes are to be assigned their IP addresses is known in advance. Purely for the sake of illustration, we considered the network to be a private IP version 4 network using either 8 bits or 16 bits for node addresses whilst the remainder of the bits are reserved for the network identifier. However, the proposed solution is equally applicable to networks using the IPv6 address space. As the network is stand alone, it is not necessary not to configure it with the address of a default gateway.

Simulation environment and parameters are described in section 5.2. Various simulation parameters such as the number of nodes, node arrival rate, network area and the number of available addresses were varied in order to gain a comprehensive analysis. In Section 5.3 we present the experiments performed and the results that were obtained. Each experiment was performed ten times and the average values were used for analysis. We compared the Wise-DAD with Strong-DAD protocol (Perkins et al, 2001). Graphical and theoretical techniques were used for analysis. In Section 5.4 we presented the conclusion of this chapter.

5.2 Simulation Environment

Wise-DAD was simulated in version 2.31 of the Network Simulator-2 tool (http://www.isi.edu/nsnam/ns) running on Ubuntu Linux 7.04 operating system with CMU extension of ns-2 to support ad-hoc networks. We also simulated another stateless protocol, Strong-DAD (Perkins et al 2001) for comparison with Wise-DAD. The following metrics were chosen to evaluate the relative performances of Wise-DAD and Strong-DAD protocols:

Latency : This refers to the average time taken for a node to be assigned an IP address. The address assignment process must be completed as quickly as possible.

Communication Overhead: Because every successfully received message, must have consumed bandwidth and power as well (Spohn and Garcia-Luna-Aceves, 2006), we use it as an evaluation metric for communication overhead. A good scheme should use as few messages as possible and the communication should be preferably local.

Address conflicts : Address uniqueness is a requirement that cannot be compromised due to the fact that node identification becomes difficult if duplicate IP addresses exist. A good scheme should minimize the probability of having more than one node using the same IP address.

In the following subsections we describe the models of the various layers of the IEEE 802.11 protocol stack that were used in this simulation.

5.2.1 Routing Protocol

Nodes were configured to use the Dynamic Source Routing (DSR) protocol. However, both Wise-DAD and Strong-DAD are independent from the routing protocol used. We did not perform simulations in which nodes transfer data coming from the application layer because we focused our attention on assessing the traffic generated by the two protocols independently from upper layers. Both implementations of Strong-DAD and Wise-DAD have no assumptions on the underlying routing protocols, because both multi-hop broadcast and one-hop broadcast were implemented without the aid of routing protocols. To verify the correctness of broadcast (both multi-hop and one-hop) implementation, we first ran the simulation for 3, 5 and 10 nodes separately. The area size was chosen to make all the nodes connected in the topology. The results show that both multi-hop and one-hop broadcast were correctly implemented in both strong and Wise-DAD.

5.2.2 Physical Data Link Layer Model

Nodes were configured to use omni-directional antennas. An omni-directional antenna radiates or receives equally well in all directions. It is also called the "non-directional" antenna because it does not favour any particular direction. This type of pattern is commonly associated with verticals, ground planes and other antenna types in which the radiator element is vertical with respect to the Earth's surface. For transmitters, the radiated signal has the same strength in all directions. This pattern is useful for broadcasting a signal to all directions or when listening for signals from all directions.

5.2.3 Medium Access Control

The link layer model used in the simulation is based on the IEEE 802.11 MAC protocol. The 802.11 family uses a MAC layer known as CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance). CSMA/CA is, like all Ethernet protocols, peer-to-peer i.e. there is no requirement for a central node. In CSMA/CA a wireless node that wants to transmit data packets performs the following sequence of steps:

i. Listen on the desired channel.

ii. If channel is idle (no active transmitters) it sends a packet.

iii. If channel is busy the node waits until transmission stops then waits again for a further contention period. The Contention period is a random period after every transmission and statistically allows every node equal access to the media.

iv. If the channel is still idle at the end of the contention period the node transmits its packet otherwise it repeats the previous step until it senses a free channel.

5.2.4 Packet Buffering Model

Every wireless multi-hop network node in the simulation used a buffer for both data and control packets that are awaiting transmission. The buffer was able to hold not more that 50 packets and implemented the drop-tail queue management algorithm. In this type of buffer, packets are transmitted on the first come first served basis. If the buffer is full, new packets are dropped.

5.3 Experiments

In this section we briefly describe experiments together with results for three experiments conducted. Section 5.3.1 presents experiments to test the scalability of our approach. In section 5.3.2 we test the robustness of Wise-DAD by varying the inter-arrival rate of nodes. Section 5.3.3 presents experiments to test the effect of interference on our protocol.

5.3.1 Experiment I: Effect of network size on Wise-DAD

The aim of this experiment was to analyze the performance of Wise-DAD on different network sizes. The idea was to test the scalability of our protocol. Only one node was configured before starting the simulations and node positions were randomly created by the simulator. We selected scenarios where destination unreachable was zero. This meant that every node could always communicate with the others during the entire simulation time. There were no node departures for the duration of the simulation since the aim of the experiment was to investigate the performance of Wise-DAD as network size increases. Table 5.1 shows the other simulation parameters for the experiment and Figure 5.1 shows the address allocation process of a 60-node network on a 1000m x 1000m area.

Table 5.1: Simulation parameters for experiment I

Parameters	**Environment**
Number of nodes	30, 40, 50, … 130
Preconfigured nodes	1
Area	1000m x 1000m
Simulation time	6000 seconds
Routing Protocol	DSR
Node arrival rate	I node / 25 seconds
Recorded parameters	Latency , number of received packets, number of address conflicts
Address range	256 , 65536

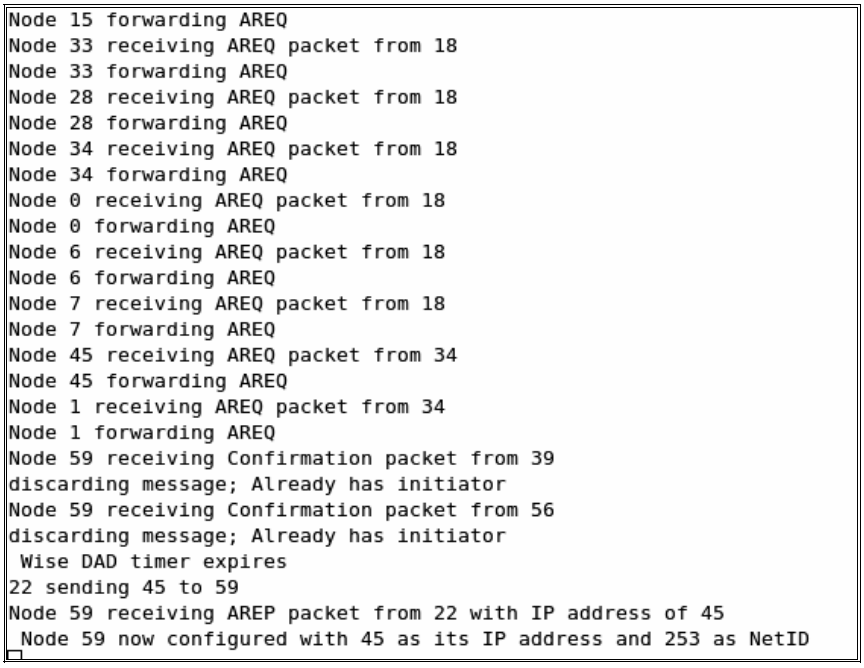

Figure 5.1: Wise-DAD simulation screenshot

(a) Effect of network size on communication overhead

Table 5.2 shows the effect of network size on communication overhead using 8-bit and 16-bit address spaces.

Table 5. 2: Effect of network size on communication overhead

Network Size	Communication Overhead (Number of Packets)			
	8-bit Address space		16-bit Address space	
	Wise-DAD	Strong-DAD	Wise-DAD	Strong-DAD
30	344	472	403	486
40	689	664	708	908
50	1161	1464	1151	1300
60	1684	2354	1560	2065
70	2600	3179	2282	3070
80	2878	4100	2910	3876
90	3337	6547	3430	6143
100	4865	7037	4675	8728
110	6150	8709	5560	9816
120	7068	12524	6074	11061
130	10156	17075	7933	13747

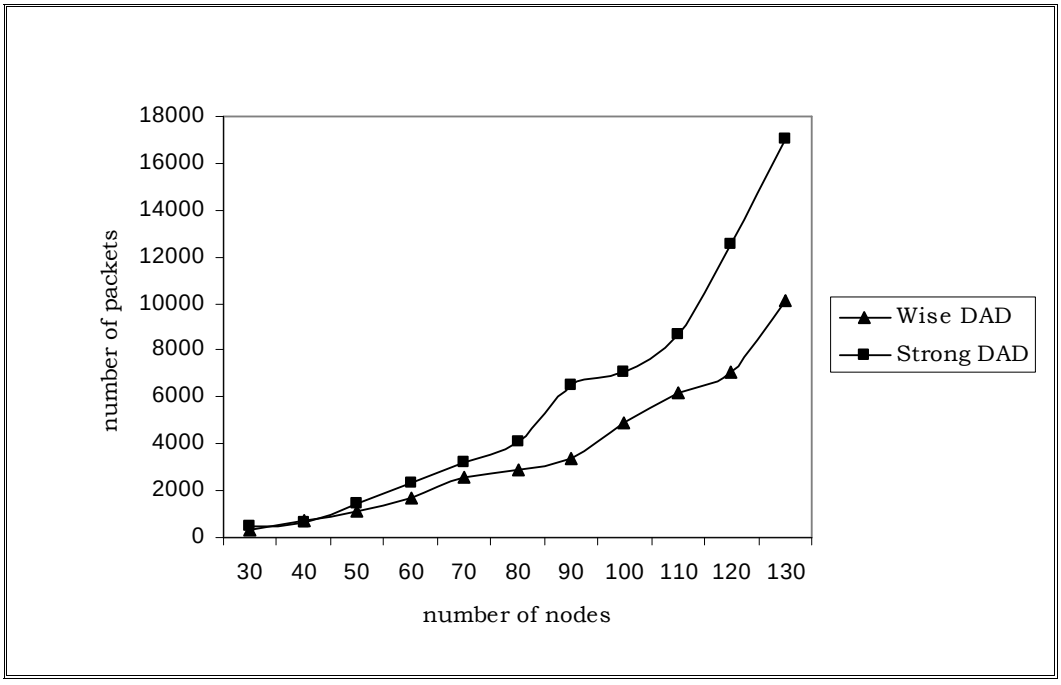

Figure 5.2: Effect of network size on communication overhead (8-bit address space)

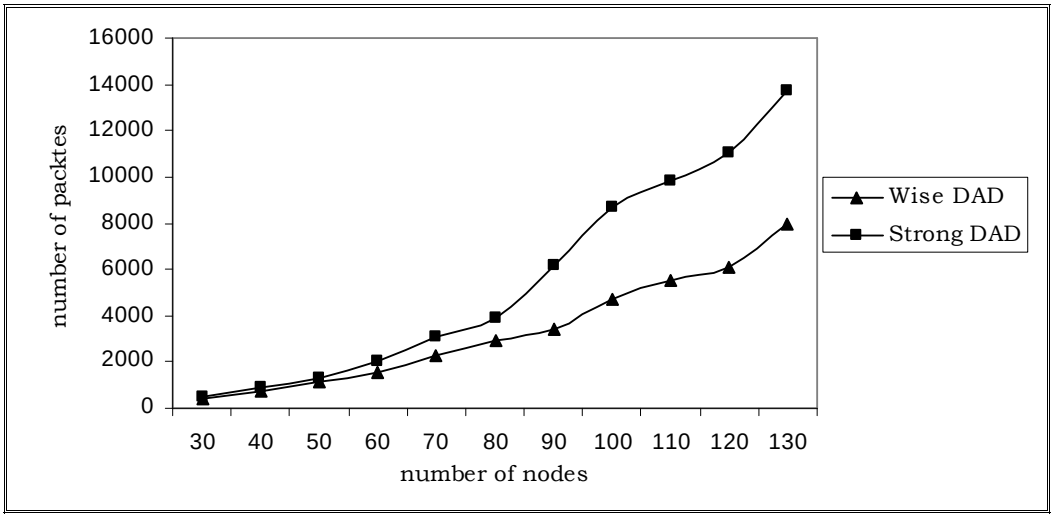

Figure 5.3: Effect of network size on communication overhead (16-bit address space)

Figure 5.3 shows the number of packet transmissions according to the number of nodes using an 8-bit address space and Figure 5.4 shows the same experiment using a 16-bit address space. Using the two address spaces, the number of packets is in proportion to the number of nodes in both Strong-DAD and Wise-DAD. During a DAD procedure, Strong-DAD floods a packet three times, and Wise-DAD only floods a packet once. That is why the number of packets recorded in Strong-DAD is more than the number of packets in Wise-DAD. For example, in a 130-node network, the number of packets in Strong-DAD went up to 17 000 whilst Wise-DAD only recorded slightly below 8 000. Although both Strong-DAD and Wise-DAD repeat the DAD procedure until the node succeeds in getting a free IP address, the probability of a successful DAD in Strong-DAD is enhanced by soft state maintenance. Both Wise-DAD and Strong-DAD show slightly better results on a 16-bit address space because the number of failed DAD trails decreased due to the large address space.

Table 5.3: Effect of network size on latency

Network size	Latency (in seconds)			
	8-bit Address Space		16-bit Address Space	
	Wise-DAD	Strong-DAD	Wise-DAD	Strong-DAD
30	1.85	5.5	1.54	5.5
40	1.82	5.5	1.5	5.5
50	1.85	5.5	1.51	5.5
60	1.83	5.51	1.48	5.5
70	1.85	5.52	1.51	5.5
80	1.84	5.53	1.5	5.5
90	1.84	5.53	1.5	5.5
100	1.85	5.54	1.5	5.51
110	1.86	5.55	1.52	5.51
120	1.95	5.56	1.52	5.52
130	2.01	5.5	1.54	5.52

(b) Effect of network size on latency

Table 5.3 shows the effect of network size on latency and Figures 5.5 and 5.6 gives graphical representation of the results. The timeout to find an initiator for Wise-DAD was set at 0.5 seconds; and the timeout for a DAD procedure was 1.8 seconds, which is calculated from the fact that the maximum hop count is 12 and the maximum one-hop round trip time is 0.15 seconds, thus the timeout must be at least 1.8 seconds (Kim et al, 2007). Strong-DAD took at least 5.4 seconds to obtain an address because it performed DAD procedure at least three times. However, this was not the case with Wise-DAD. The number of DAD trials was barely more than. This is attributed to state information maintenance in which helps to reduce the possibilities of failed DAD thereby reducing both the latency and communication overhead. State information is updated as nodes join the network.

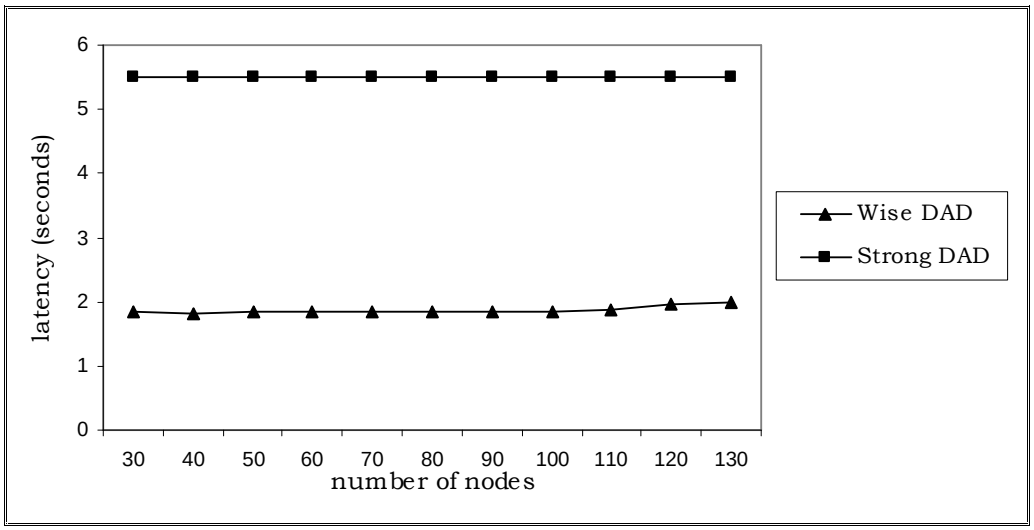

Figure 5.4: Effect of network size on latency (8-bit address space)

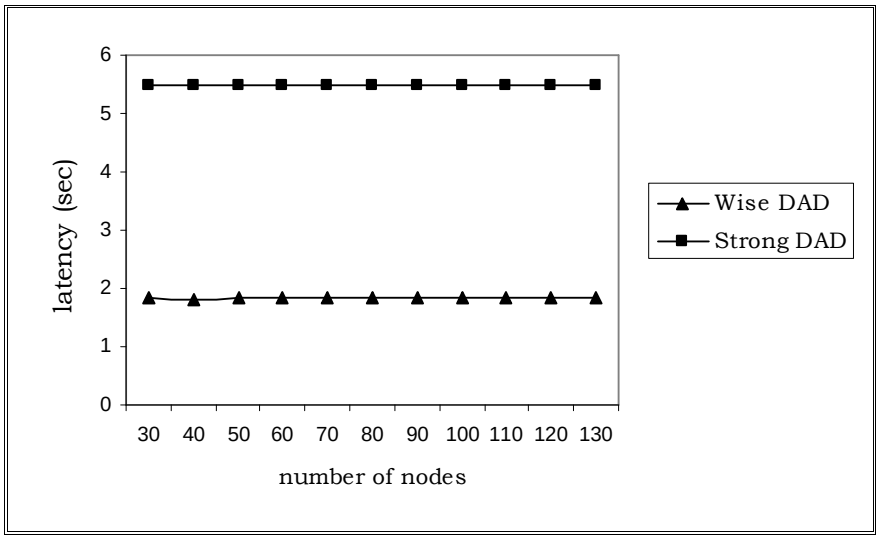

Figure 5.5: Effect of network size on latency (16-bit address space)

(c) Effect of network size on IP address conflicts

One of the most important requirements in IP address auto-configuration is to guarantee the uniqueness of allocated addresses, since address conflicts may cause abnormal behavior of routing protocols and applications (Kim et al, 2007). Fig. 5.7 and 5.8 shows the number of conflicting IP addresses according to the number of nodes. The number of IP address conflicts in both Wise-DAD and Strong-DAD increases with network size because of the fact that the address space is a finite domain hence the probability of getting a free IP address decreases as network size increase. The other reason could be message losses caused by MAC collisions as traffic increases (due to increase in network size) on the network.

Using a 16-bit address space, the number of address conflicts decreased dramatically in both strong and Wise-DAD because of the large address space. The probability of generating the same address more than once is very low. For example, with 100 nodes the probability is as low as 0.0015. In simulation, however, this probability might not be the same since computer generated random numbers are not purely random. The numbers are generated from certain algorithms that follow a certain sequence which might even include generating certain numbers more than others. Wise-DAD shows relatively better results than Strong-DAD on both 8-bit and 16-bit address spaces because of state information maintenance.

Table 5.4: Effect on network size on address conflicts

Network Size	Number of Duplicate IP addresses			
	8- bit Address Space		16-bit Address Space	
	Wise-DAD	Strong-DAD	Wise-DAD	Strong-DAD
30	0	1	0	0
40	1.9	2	0	0
50	1	3.4	0.15	1
60	2	6.2	0.18	1.28
70	2	7	0.24	1.25
80	2	7.5	0.18	1.42
90	3	9	0.21	1.55
100	4	14	0.43	2
110	4	20	1	2.16
120	5	22	1	3
130	5	30	1.06	3.23

Generally, in strong-DAD the probability, P, of getting a free IP address on a given DAD trial is:

$$P = \frac{S - N}{S}$$

Where S is the number of possible IP addresses, N is the network size. From the above equation is it clear that communication overhead increases as the network size increase since the number of DAD trials are likely to increase. This is so because as $N \to S$, P becomes very small. However, since Wise-DAD maintains state information, this probability is reduced. For example, if the state information is L then the probability of address conflicts is given by:

$$P = \frac{S - N}{S - L}.$$

As nodes join the network both N and L increase (although not at the same rate) hence the P does not increase rapidly.

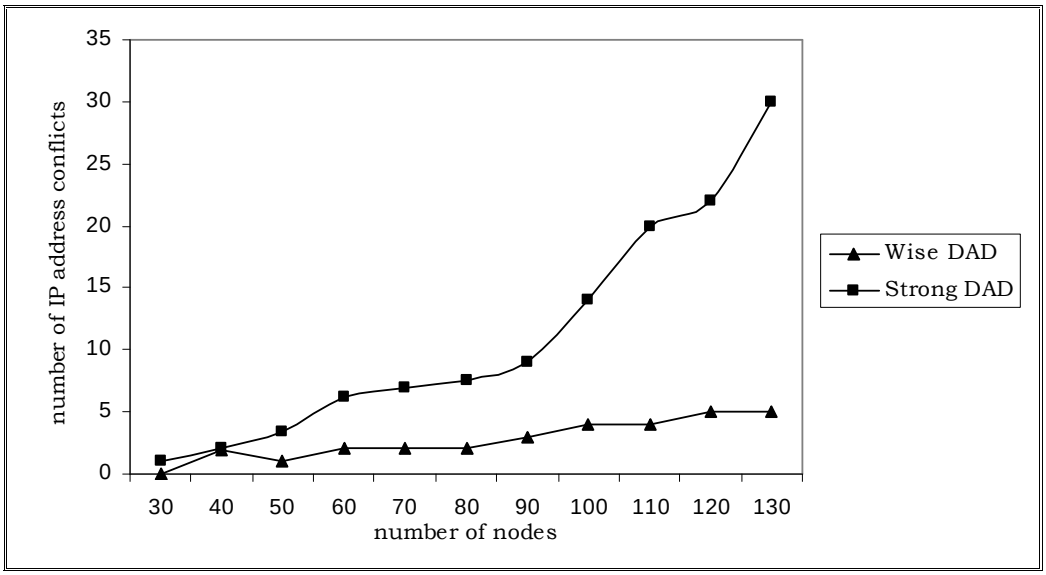

Figure 5.6: Effect of network size on address conflicts (8-bit address space)

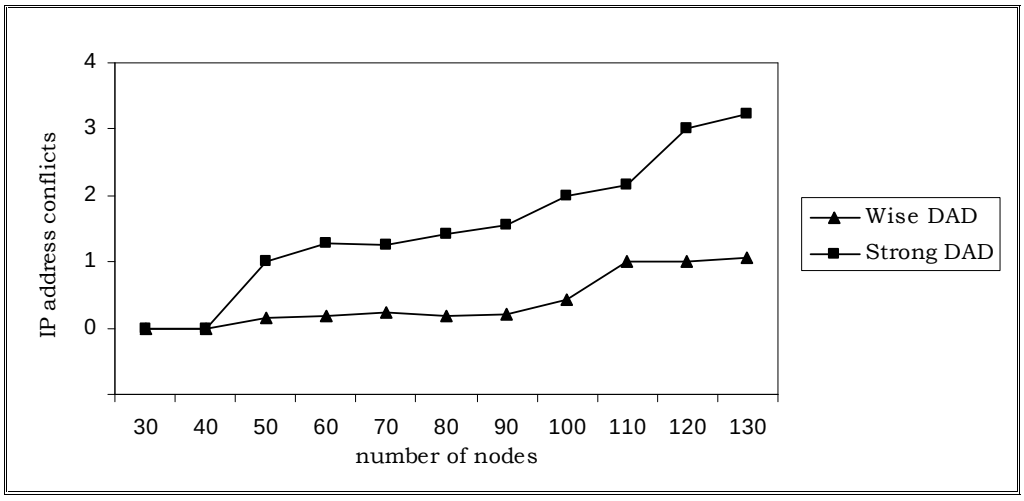

Figure 5.7: Effect of network size on address conflicts (16-bit address space)

Table 5.4 shows the effect of network size on address conflicts and Figures 5.7 and 5.8 gives graphical representation of these results.

Table 5.5: Simulation parameters for experiment II

Parameters	**Environment**
Number of nodes	80
Preconfigured nodes	1
Area	1000m x 1000m
Simulation time	6000 seconds
Routing Protocol	DSR
Node arrival rate	1 node every 5, 10, 15, 20, 25, 30 seconds
Recorded parameters	Latency , number of packets, number of address conflicts
Address range	8-bit address space

5.3.2 Experiment II: Effect of node arrival rate on Wise-DAD

To test the robustness of Wise-DAD, we varied inter-arrival time of the nodes. An 8-bit address space was used, and the number of nodes was fixed at 80 (the median of experiment I). Address conflicts, communication overhead, and address allocation latency for both Strong-DAD and Wise-DAD were recorded. During the entire duration of the simulation, every node could always communicate with the others. Table 5.5 shows the other simulation parameters used in the experiment.

(a) Effect of node arrival rate on address conflicts

Table 5.6 shows the effect of node arrival rate on IP address conflicts and the results of this experiment are depicted graphically in Figure 5.9

Table 5. 6: Effect of node arrival rate on address uniqueness

Node Arrival Rate	Number of IP address conflicts	
	Wise-DAD	**Strong-DAD**
1 node/ 5 seconds	1	15
1 node/ 10 seconds	1	13
1 node/ 15 seconds	1	10
1 node/ 20 seconds	2	10
1 node/ 25 seconds	2	9
1 node/ 30 seconds	1	6

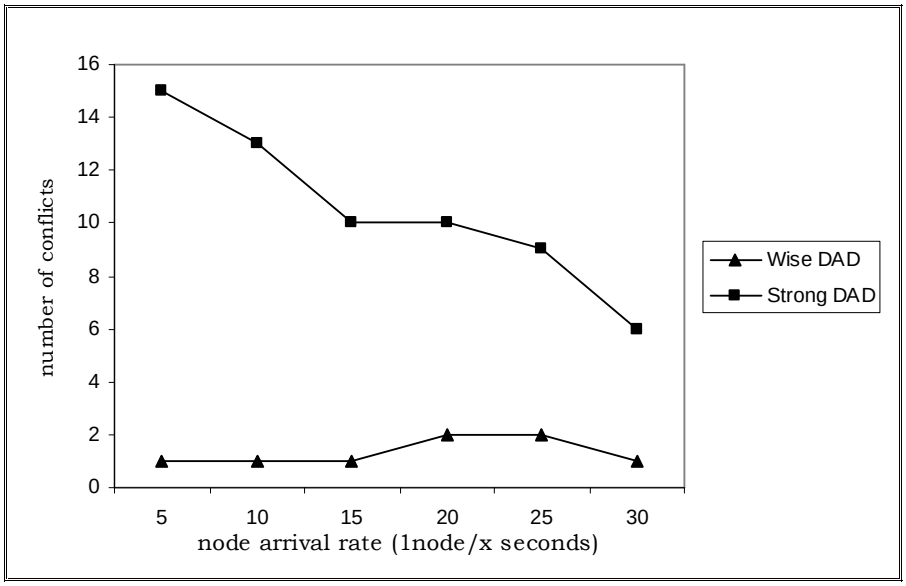

Figure 5.8: Effect of node arrival rate on address uniqueness

Figure 5.9 shows that Wise-DAD did not show significant changes in IP address conflicts as node arrival rate was varied. The number of conflicts ranged between 0 and 2. This is due to the fact that node admission helps in updating state information. Address conflicts in Strong-DAD on the other hand, was inversely proportional to node arrival rate. This is attributed to the fact the Strong-DAD does not provide a mechanism to resolve a situation whereby two or more nodes are requesting for the same IP address at the same time. High node arrival rate are likely

to result in such a situation. Message losses are likely to be another cause for high address conflicts when node arrival rate was high. However, Figure 5.10 shows that the total number of packets received by nodes is not significantly affected by the node arrival rate. Therefore, message delays not losses could have caused address conflicts in Strong-DAD.

(b) Effect of node arrival rate on communication overhead

Table 5.7 shows the effect of node arrival rate on communication overhead and the results of this experiment are depicted graphically in Figure 5.10. The graph shows that the number of packets in both Wise-DAD and Strong-DAD is not affected by the rate at which nodes join the network. Although the number of packets is not constant, the variations are not sufficient to suggest that node arrival rate has an effect (positive or negative) on the traffic generated by both schemes. The number of packets sent by an initiator during an IP address assignment process depend on the success of DAD. Node arrival does not have an effect on the success of DAD hence communication overhead did not change as the node arrival rate was varied. This implies that wise-DAD is a robust protocol and its performance is not affected by network dynamics.

Table 5.7: Effect of node arrival rate on communication overhead

	Communication overhead (Number of packets)	
Node Arrival Rate	**Wise-DAD**	**Strong-DAD**
1 node/ 5 seconds	3070	3650
1 node/ 10 seconds	3219	3680
1 node/ 15 seconds	3320	3970
1 node/ 20 seconds	3272	3871
1 node/ 25 seconds	3420	3830
1 node/ 30 seconds	3140	3900

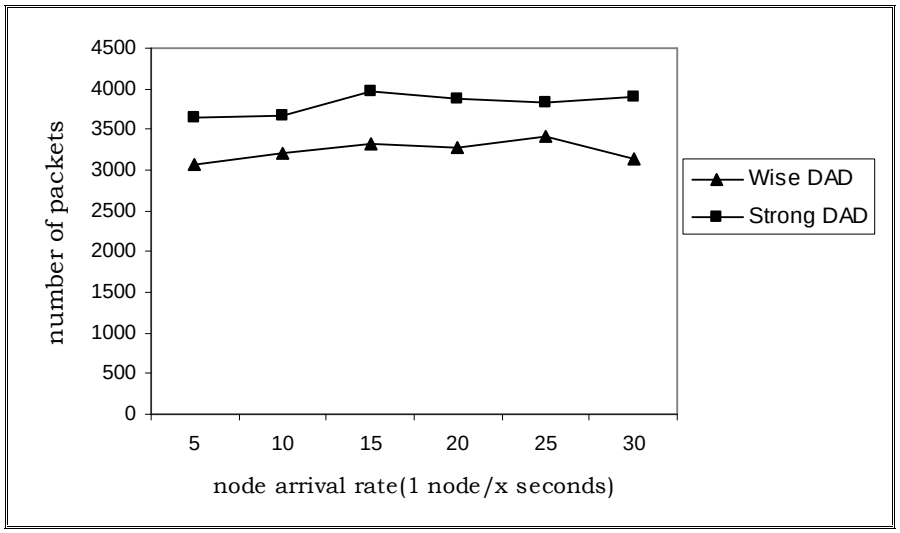

Figure 5.9: Effect of node arrival rate on communication overhead

(c) Effect of node arrival rate on latency

Table 5.8 shows the effect of node arrival rate on latency and Figure 5.11 gives a graphical representation of the results.

Table 5.8: Effect of node arrival rate on latency

Node Arrival Rate	Latency (in seconds)	
	Wise-DAD	**Strong-DAD**
1 node/ 5 seconds	1.501	5.5
1 node/ 10 seconds	1.5	5.501
1 node/ 15 seconds	1.5	5.5
1 node/ 20 seconds	1.5	5.51
1 node/ 25 seconds	1.5	5.5
1 node/ 30 seconds	1.51	5.503

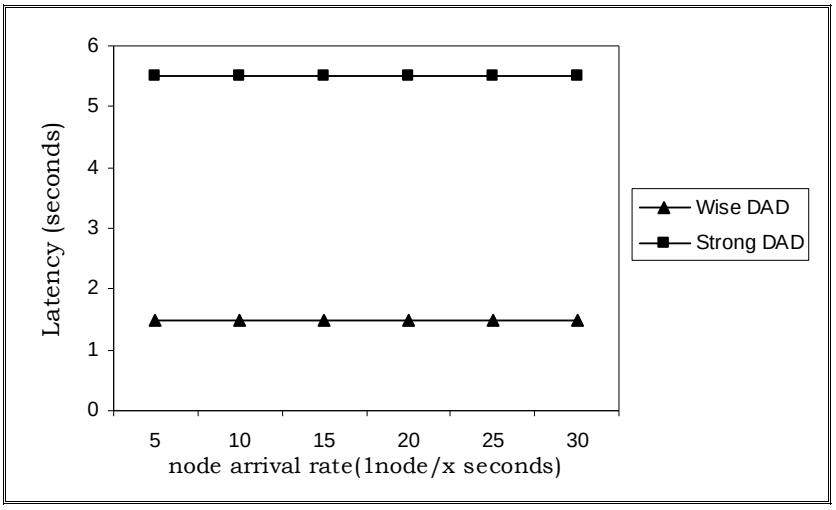

Figure 5.10: Effect of node arrival rate on latency

In stateless approaches, address allocation latency is affected by the number of DAD trials and the DAD timeout period. Figure 5.10 shows that node arrival rate does not have an effect on communication overhead hence the DAD trials were not affected by node arrival rate. As a result the latency shown in Figure 5.11 did not change as the node arrival rate was varied. This again shows Wise-DAD's robustness in the face of network dynamics.

5.3.3 Experiment III: Effect of interference on Wise-DAD

This experiment was performed to assess the impact of interference on the performance of Wise-DAD. To show the impact of interference on address uniqueness and communication overhead, we varied node density by changing the simulation area. Node positions were generated at random hence we assume the average node density decreases as the area increase. However, the actual node degree was not recorded. Node density affects interference which in-turn affects message delivery (Mudali et al, 2007). Messages from neighbours are received free of errors provided only one neighbour is transmitting (Borbash et al, 2007). An 8-bit address space was used, whilst the number of nodes was fixed at 70. Address uniqueness and communication overhead for both Strong-DAD and Wise-DAD were recorded. Figure 5.11 and Figure 5.12 shows how 70 nodes are spaced on $600m^2$ (high interference) and $1200m^2$ (low interference) respectively.

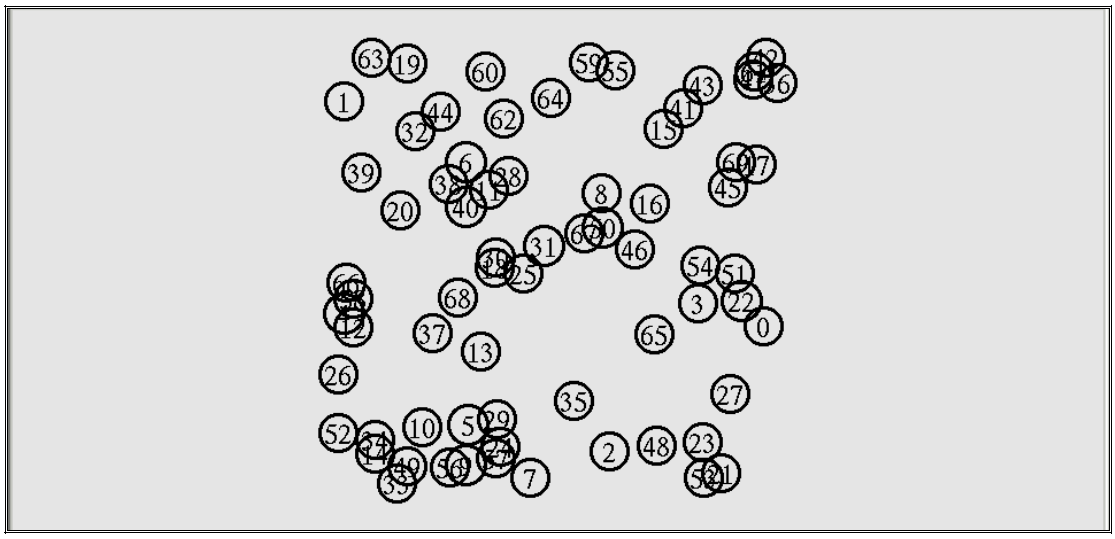

Figure 5.11: Network with high interference

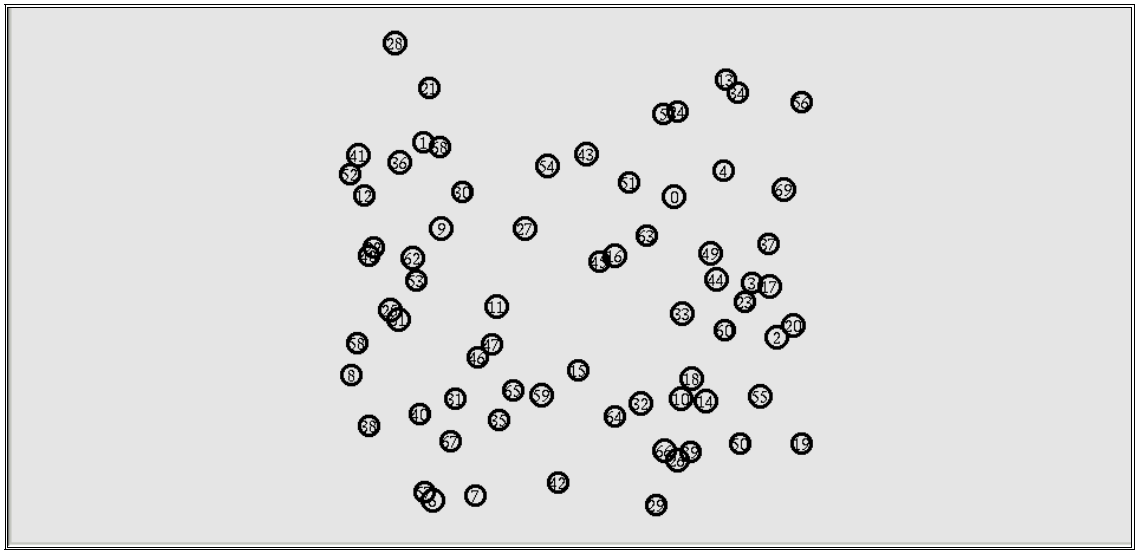

Figure 5.12: Network with low interference

Table 5.9 shows the other simulation parameters. Like in experiment I and experiment II, we selected scenarios where every node could always communicate with the others during the entire simulation time. This was done to make sure that the node density is always constant for the duration of the simulation. Also because of the same reason there were no node departures and mobility for the entire duration of the simulation.

Table 5.9: Simulation parameters for experiment III

Parameters	**Environment**
Number of nodes	70
Preconfigured nodes	1
Area	(500m, 600,700, … 1200m)^2
Simulation time	6000 seconds
Routing Protocol	DSR
Node arrival rate	25 seconds
Recorded parameters	Latency , number of packets, number of address conflicts
Address range	256

Table 5.10: Effect of interference on address uniqueness

Area (m^2)	**Number of address conflicts**	
	Wise-DAD	**Strong-DAD**
500	0	3
600	0	6
700	0	7
800	2	8
900	2	3
1000	2	3
1100	3	6
1200	5	9

(a) Effect of interference on IP address conflicts

Table 5.10 shows the effect of node density on address uniqueness and the results of the experiment are depicted graphically in Figure 5.14.

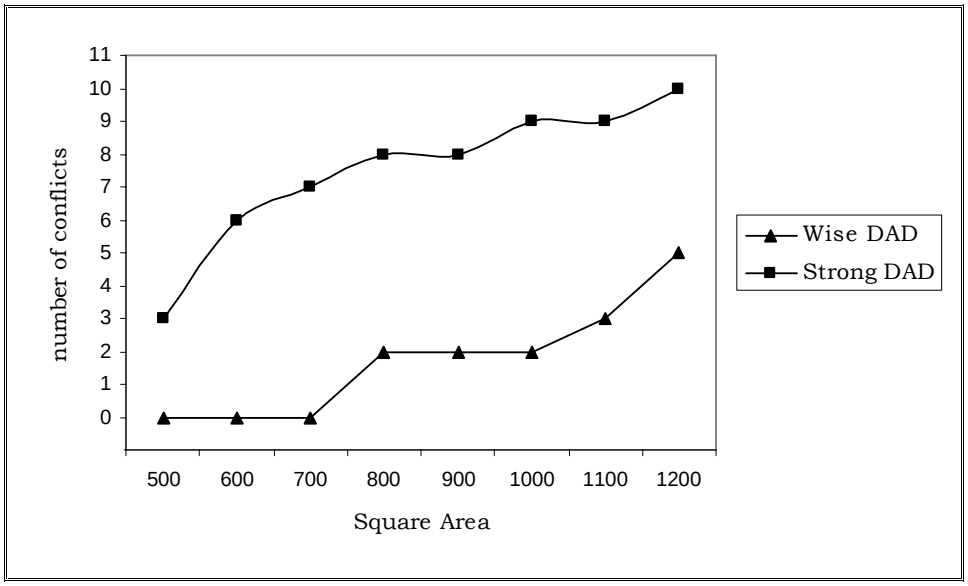

Figure 5.13: Effect of node density on address uniqueness

In both Strong-DAD and Wise-DAD, the number of IP address conflicts is proportional to the area size i.e. inversely proportional to node density. This is due to the fact that message delivery is better in high node density due to redundant routes. However, interference is also high if the node density is high. Because of state maintenance, the number of conflicts in Wise-DAD is relatively lower than in Strong-DAD.

(b) Effect of interference on communication overhead

Table 5.11 shows the effect of interference on communication overhead and the results of this experiment are depicted graphically in Figure 5.15

Table 5.11: Effect of interference on communication overhead

Area (m^2)	Number of packets	
	Wise-DAD	**Strong-DAD**
500	2280	16154
600	2280	10189
700	2353	6874
800	2390	6159
900	2342	4539
1000	2469	2666
1100	2384	2634
1200	3326	1848

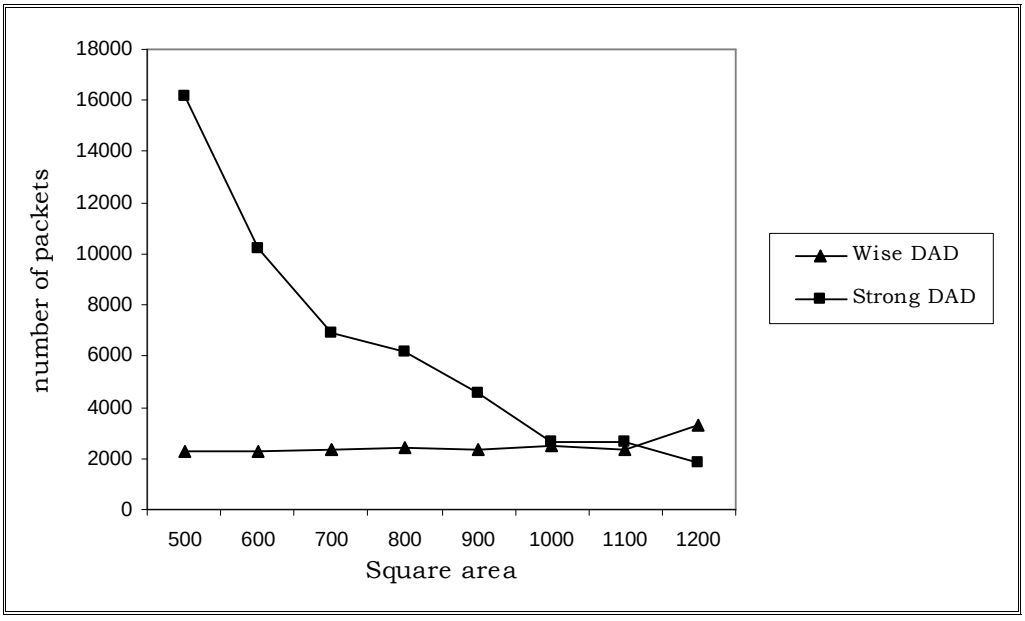

Figure 5.14: Effect of interference on communication overhead

The number of packets recorded in Wise-DAD was not affected by node density, but however Strong-DAD had high communication overhead when the node density was high. Figure 5.14 shows that as the node density decreases, the number of IP address duplicates increases (for Strong-DAD) whilst Figure 5.15 show a decrease in communication overhead as the node density decreases. A decrease in communication overhead in Strong-DAD was due to the fact that as node density decrease, the link number decreases as well. As the link number decreases, some nodes won't be able to defend their IP addresses during a DAD process hence an increase in the number of conflicts (Figure 5.14).

5.4 Summary

In this chapter we presented the simulation results of the Wise-DAD protocol. We compared Wise-DAD with another stateless protocol, Strong-DAD. Experiments on the effect of network size, interference and node arrival rate on communication overhead, address uniqueness and latency were done. The goal of was to simultaneously achieve low latency, low communication overhead and low address conflicts. This goal was achieved by the introduction of passively updated state information. Simulation results show that Wise-DAD outperforms Strong-DAD in all the three metrics used for performance evaluation.

CHAPTER SIX

CONCLUSION AND FUTURE WORK

6.1 Introduction

This study is a successful attempt to investigate automatic IP addressing techniques. Lack of manual management in ad-hoc networks means that automatic configuration is highly desirable. Automatic configuration of nodes in wireless ad hoc network will help in reducing administration efforts by users and network administrators. Our initial investigation into this area identified the need for achieving high levels of address uniqueness without compromising on latency and communication overhead. The aim of this investigation was captured in the following research questions: (1) how can we achieve high levels of address uniqueness without compromising on latency and communication overhead? (2) Which IP address auto-configuration approach or paradigm can yield an optimal protocol performance in terms of low latency, low communication overhead and minimal address conflicts?
Addressing these issues will, among other things, help in providing QoS guarantees in the network. Reducing communication overhead and improving on address uniqueness will aid higher level protocols to function well thereby improving Quality of Service (QoS) provisioning.

In order to achieve our overall goal and provide answers to our research questions, the following objectives were set: (1) To investigate the existing IP addressing protocols, paying attention at their effect on communication overhead, latency and address uniqueness, (2) To develop an IP address auto-configuration scheme, and (3)

To evaluate the performance of the proposed scheme against existing schemes and show improvements achieved through simulation.

This chapter gives an overview of the work presented in this dissertation. Section 6.2 gives a summary of this work whilst section 6.3 gives a critical analysis of the limitations of the Wise-DAD protocol. Possible enhancements to our work are also given in the same section.

6.2 Summary and Conclusion

An investigation that was conducted concluded that it would be difficult to develop a method that can guarantee address uniqueness without performing a DAD procedure. This effectively answered our second research question. The Wise-DAD protocol was proposed in chapter 4 to achieve our goal. We achieved our goal by introducing state information maintenance which is passively collected and not actively synchronized. Passively collecting state information reduces the number of DAD trials thereby reducing latency. This provided an answer to our first research question.

Simulation experiments were done in NS-2 to test the performance of the Wise-DAD protocol. A comparative analysis was then conducted. The Wise-DAD scheme was compared with the Strong-DAD protocol. We conducted experiments to investigate the effect of network size, interference and node arrival rate on communication overhead, address uniqueness and latency. The results obtained from the simulation experiments showed that Wise-DAD outperforms Strong-DAD in all the three metrics used for performance evaluation. The results obtained are summarized as in the following paragraph.

First, Wise-DAD showed better scalability since it performed better than Strong-DAD when network size was increased. Communication overhead in Wise-DAD was generally low whilst the latency was generally uniform. The number of IP address duplicates recorded was reasonably low. For example a 130-node network using an 8-bit address space recorded an average of 5 address conflicts. The network size was varied from 30 to 130. Second, Wise-DAD was not affected by node arrival rate on all the three metrics that were recorded. On the other hand, the number of address duplicates in Strong-DAD decreased as the node arrival rate was increased. Interference significantly affected communication overhead recorded in Strong-DAD. Wise-DAD, on the other hand, was not affected by interference. The number of address conflicts in both protocols showed an inverse relationship to interference. However, the number of conflicts for both protocols was significantly different; Wise-DAD recorded much less address conflicts than Strong-DAD.

In designing these protocols, the following factors must be considered [4]: (1) Network partitioning and merging; (2) Duplicate Address Detection (DAD); (3) Scalability; (4) Security and Authentication. There are various conclusions regarding the use of state information that can be drawn from this work. First, the inclusion of soft state information can greatly reduce latency, the number of DAD trials and IP address conflicts. We can therefore unambiguously conclude that the introduction of state information can increase the scalability of IP address auto-configuration schemes. With minor modifications, state information can be used to detect duplicate addresses. The use of a key-address combination such as the one used in Weak-DAD can be added to the state information for the detection of duplicate IP addresses.

6.3 Limitations and Future Enhancements

In this section we outline some shortcomings of our proposal and point out future focus of this work. Some of the concerns given in this section are not directly related to our problem statement or research questions, but their importance to IP address auto-configuration cannot be ignored. Although the Wise-DAD protocol will enhance the capabilities of future ad-hoc networks, problems such as security still need to be completely worked out. The memory and processing requirements of our solution is another important issue needing further investigation. For successful application of Wise-DAD protocol in the pervasive environment, the protocol must be lightweight enough to be deployed in handheld and other capacity-constrained devices.

Real life experiments still need to be conducted in order to come up with realistic results. Ideally a real life test-bed should be used to experiment with the ideas reported in this dissertation. However, to obtain any meaningful results for scenarios such as network merging, it requires a fairly large number of participating nodes in the network which was not possible due to both budget and time constraints. Therefore, in this work we used simulations only to test the performance of our solution. A test-bed being constructed at the University of Zululand will be used and results compared with the ones obtained from our simulation. Network merging strategy that was used in Wise-DAD still need more rigorous testing since it employs broadcasting. A passive approach of detecting address conflicts looks promising, because it does not consume any additional bandwidth. Although not the focus of this work, we need a simulation study for different network merger cases.

Security will be the major future focus of this work since it was not considered in current study. Wise-DAD did not consider possible existence of malicious nodes. All nodes were assumed to be authentic although this assumption is not realistic. Every node that requested for an IP address was only denied if the IP addresses are not available. No security mechanisms were employed. Attacks on networks are becoming increasingly intelligent and more fatal (Kim et al 2007), therefore, address configuration schemes need a concrete solution for security.

No address resolution mechanism was proposed. Wise-DAD assumes that all nodes will be configured with unique IP addresses. However, from the results obtained, it was observed that this is not always the case; hence future work should focus on detecting IP address conflicts which are not related to network merging.

Although all nodes were running DSR routing protocol, we did not perform simulations in which nodes transfer data coming from the application layer, because we focused our attention on assessing the traffic generated by Wise-DAD independently from upper layers. However, to test our idea in realistic situations, application layer traffic must be present. Future work will also focus on this aspect in order to obtain meaningful results.

Purely for the sake of illustration, we considered the network to be a private IP version 4 network using either 8 bits or 16 bits for node addresses whilst the remainder of the bits are reserved for the network identifier. However, the proposed solution is equally applicable to networks using the IPv6 address space. As we are drifting towards IPv6, it could have been better if this work was tested in IPv6.

BIBLIOGRAPHY

Bernados, C. and Calderon, M. (2005). "Survey of IP address autoconfiguration mechanisms for MANETs ", draft-bernardos-manet-autoconf-survey-03, 2005.

Borbash, S., Ephremides, A., McGlynn, M.J. (2007). "An asynchronous neighbour discovery algorithm for wireless sensor networks", Ad-hoc Networks, Volume 5, Issue 7, pp: 998-1016, September 2007.

Campos, R. and Ricardo, M. (2005). "Dynamic Autoconfiguration in 4G Networks: Problem Statement and Preliminary Solution", in: the Proceedings of the 1st ACM workshop on Dynamic interconnection of networks, 2005.

Cavalli, A. and Orset, J. (2005). "Secure hosts auto-configuration in mobile ad-hoc networks", Ad-hoc Networks, Volume 3, Issue 5, pp: 656-667, September 2005.

Crichton M, "Prey", (2002). New York : HarperCollins, ISBN: 0066214122, 2002.

Dijkstra, F., Van der Ham, J., Cees, T.A.M. (2006). "Using zero configuration technology for IP addressing in optical networks", Future Generation Computer Systems, Volume 22, Issue 8, pp 908-914, October 2006.

Droms, R. (1997). "Dynamic host configuration protocol", Network Working Group RFC 2131, March 1997.

Fall, K, and Varadhan, K. (2008). "The ns Manual--the VINT Project", April 2008. Available: http://www.isi.edu/nsnam/ns/ns-documentation.html

Fan, Z. and Subramani, S. (2005). "An address autoconfiguration protocol for IPv6 hosts in a mobile adhoc Network", Computer Communications, Volume 28, Issue 4, pp: 339-350, March 2005.

Fazio, M., Villari, M., Puliafito, A. (2006). "AIPAC: Automatic IP address configuration in mobile ad-hoc networks" ,Computer Communications, Volume 29, Issue 8, pp: 1189-1200, May 2006.

Gao, Z., Wang, L., Yang, M., Yang, X. (2006). "CNPGSDP: An efficient group-based service discovery protocol for MANETs" , Computer Networks, Volume 50, Issue 16, pp: 3165-3182, 14 November 2006.

Günes, M. and Reibel, J. (2002). "An IP Address Configuration Algorithm for Zeroconf Mobile Multihop Ad-hoc Networks," Proceedings of the International Workshop on Broadband Wireless Ad-Hoc Networks and Services, Sophia Antipolis, France, Sept. 2002.

Indrasinghe, S., Pereira, R., Mokhtar, H. (2006). "Hosts Address Auto Configuration for Mobile Ad-hoc Networks" , in the proceedings of HET-NETs, West Yorkshire UK, 2006.

Jeong, J., Park, J., Kim, H., Kim, D. (2004). "Ad-hoc IP Address Autoconfiguration for AODV", draft-jeong-manet-aodv-addr-autoconf-01.txt, January 2004.

Kim, N., Ahn, S., Lee, Y. (2007). "AROD: An Address Autoconfiguration with Address Reservation and Optimistic Duplicated Address Detection for Mobile Ad-hoc Networks", Computer Communications, Volume 30, Issue 8, pp: 1913-1925, June 2007.

Mohsin, M. and Prakash, R. (2002). "IP Address Assignment in a Mobile Ad-hoc Network," Proceedings of IEEE MILCOM 2002, Anaheim, CA, Oct. 2002.

Mudali, P., Nyandeni, T.C., Adigun, M.O. (2007). "A Performance comparison of Wireless Multi-Hop Network Topologies Based on Average Node Degree", in the proceedings of Southern Africa Telecommunication Networks and Applications Conference, 19 September 2007.

Mutanga, M.B., Nyandeni, T.C., Mudali, P., Xulu, S.S., Adigun, M.O. (2008). "Wise-DAD Auto-Configuration for Wireless Multi-hop Networks", In the proceedings of Southern Africa Telecommunication Networks and Applications Conference", 7 -10 Sep 2008.

Nesargi, S. and Prakash, R. (2002). "MANETconf: configuration of hosts in a mobile ad-hoc Network", in: Proceedings of the 21st Annual Joint Conference of IEEE Computer and Communication Societies (INFOCOM 2002), New York, June 2002.

Perkins, C., Malinen, T., Wakikawa, R., Belding-Royer, E., Sun, Y. (2001). "IP address autoconfiguration for ad-hoc networks", IETF Internet Draft 2001.

Prehofer, C. and Bettstetter, C. (2005). "Self-Organization in Communication Networks: Principles and Design Paradigms", IEEE Communications Magazine Volume 43, Issue 7pp: 78 - 85, July 2005.

Saxena, N., Tsudik, G., Yi, J.H. (2005). "Efficient Node Admission for Short-lived Mobile Ad-hoc Networks", Proceedings of the 13TH IEEE International Conference on Network Protocols, pp: 269 – 278, 2005.

Spohn, M.A. and Garcia-Luna-Aceves, J.J. (2006). "Improving route discovery in on-demand routing protocols using two-hop connected dominating sets", Ad-hoc Networks, Volume 4, Issue 4, pp: 509-531, July 2006.

Sun, Y. and Belding-Royer, E.M. (2003). "Dynamic Address Configuration in Mobile AdHoc Networks," UCSB tech. rep. 2003-11, Santa Barbara, CA, June 2003.

Sun, Y., Belding-Royer, E.M., Perkins, C.E. (2002). "Internet Connectivity for Ad-hoc Mobile Networks ", International Journal of Wireless Information Networks special issue on Mobile Ad-hoc Networks (MANETs): Standards, Research, Applications", 9(2), April 2002.

Toner, S. and O'Mahony, D. (2003). "Self-Organising Node Address Management in Ad-hoc Networks", in Springer Verlag Lecture notes in Computer Science 2775, Springer Verlag, Berlin, pp: 476-483, 2003.

Vaidya, N.H. (2002). "Weak Duplicate Address Detection in Mobile Ad-hoc Networks," Proceedings of ACM MobiHoc 2002, Lausanne, Switzerland, pp: 206–216, June 2002.

Weniger, K. and Zitterbart, M. (2004). "Address Autoconfiguration in Mobile Ad-hoc Networks: Current Approaches and Future Directions", EEE Network Magazine Special issue on 'Ad-hoc networking: data communications & topology control', Jul 2004.

Weniger, K. (2005). "PACMAN: Passive Autoconfiguration for Mobile Ad-hoc Networks," IEEE J. Sel. Areas Commun., Special Issue on Wireless Ad-hoc Networks, vol.23, no.3, pp: 507–519, March 2005.

Weniger, K. (2004). " Passive Duplicate Address Detection in Mobile Ad-hoc Networks" In IEEE Wireless Communications and Networking Conference (WCNC), New Orleans, USA, March 2004.

Williams, A. (2002). "Requirements for automatic configuration of IP hosts. Zero Configuration Networking", IETF Internet Draft, 2002.

Zhou, H. (2003). "A Survey on Routing Protocols in MANETs", Technical Report: MSU-CSE-03-08, Mar 28, 2003.

Zhou, H. Ni, L. Mutka, M. (2003). "Prophet address allocation for large scale manets, Ad-hoc Networks", Volume 1, Issue 4, pp: 423-434, November 2003.

APPENDIX A: Wise-DAD Protocol Header File

```
#ifndef _WiseDAD_H_
#define _WiseDAD_H_

#include <ip.h>
#include <packet.h>
#include <random.h>
#include <timer-handler.h>
#include <agent.h>
#include <config.h>
#include <scheduler.h>
#include <vector.h>
#define MAX_RETRY 3
#define RETRY_TIMEOUT 30 // 30 ms
#define WiseDAD_PORT 224
#define CURRENT_TIME (Scheduler::instance()).clock()
// WiseDAD address allocation agent
class WiseDAD;
// A timer used by initiator to send reply
class WiseDADTimer : public TimerHandler
{
        public:
        WiseDADTimer(WiseDAD* a) : TimerHandler(), agent(a) { }
        inline virtual void expire(Event*);
        private:
        WiseDAD* agent;
};
// timers used by new node
class WiseDAD_RTJTimer : public TimerHandler
{
        public:
        WiseDAD_RTJTimer(WiseDAD* a) : TimerHandler(), agent(a) { }
        inline virtual void expire(Event*);
        private:
        WiseDAD* agent;
};
class WiseDAD_BusyTimer : public TimerHandler
{
        public:
        WiseDAD_BusyTimer(WiseDAD* a) : TimerHandler(), agent(a) { }
        inline virtual void expire(Event*);
        private:
        WiseDAD* agent;
};
class WiseDAD_HelloTimer : public TimerHandler
{
        public:
        WiseDAD_HelloTimer(WiseDAD* a) : TimerHandler(), agent(a) { }
        inline virtual void expire(Event*);
        private:
        WiseDAD* agent;
};
```

```
class WiseDAD : public Agent
{
        friend class BroadcastTimer;
        public:
        WiseDAD(void);
        void recv(Packet* p, Handler*);
        void sendAREQ(void);
        void sendAREP(nsaddr_t dest);
        void sendIPCONFLICT(nsaddr_t dest);
        void sendREQUEST_TO_JOIN(void);
        void sendCONFIRMATION(nsaddr_t dest);
        void sendSELECT_INITIATOR(nsaddr_t dest);
        void sendGOODBYE(void);
        void sendHello(void);
        void sendMerger(void);
        int counter;
        int times;
        int found;
        int dummy;
        int command(int, const char* const*);
        int conflicts;
        int configured; // Flag indicating whether the node has been configured
        u_int32_t index; // nsaddr_t index; IP address of this node
        int bid; // Broadcast ID
        int WiseDAD_index;
        int WiseDAD_IPAddress; // Address obtained from Wise DAD Allocation
        vector <int> AllocationTable;
        int ReversePath[160][2]; // used for replying
        int packetNumber, seqnumber;
        int rtable[160];
        int ActiveNodes; // number of nodes in the Allocation Table
        int HopCount; // Path Length travelled by a message
        int DAD_trials, localTrials, localDAD; // Number of DAD trials made by an initiator
        int busy;
        int state;
        double timeSent;
        int packets , localPackets;
        u_int32_t WiseDAD_nid;
        int RequestedIP;
        int Req;
        nsaddr_t MyInitiator;
        nsaddr_t Requestor;
        WiseDADTimer btimer; // Timer for AREQ
        WiseDAD_RTJTimer RTJ_Timer; // Time for Request to Join message
        WiseDAD_BusyTimer BusyTimer;
        WiseDAD_HelloTimer HelloTimer;
        double time, interval; // Timestamp used in statistics
        double packetTime;
        int addressSpace;
        double startTime, endTime; // latency
        int retries,// Use in backoff algorithm
        receives, // The number of packets received
        debug; // Flag controlling print out debug information repeats;
        // Statistics about retry times
        NsObject* ll;
};
#endif
```

APPENDIX B: Wise-DAD Simulator Code

```
#include "WiseDAD_packet.h"
#include "WiseDAD.h"
#include <vector.h>
#include <iostream>
#include <stdio.h>
int hdr_WiseDAD::offset_;
// Packet Header Class for WiseDAD address allocation

static class WiseDADHeaderClass : public PacketHeaderClass
{
public:
        WiseDADHeaderClass() : PacketHeaderClass("PacketHeader/WiseDADHeader",
        sizeof(hdr_WiseDAD))
        {
                bind_offset(&hdr_WiseDAD::offset_);
        }
} class_WiseDADhdr;

static class WiseDADClass : public TclClass
{
public:
        WiseDADClass() : TclClass("Agent/WiseDAD") { }
        TclObject* create(int argc, const char* const* argv)
        {
                return (new WiseDAD);
        }
} class_WiseDAD;

// Tcl interface for WiseDAD Allocation
int WiseDAD::command(int argc, const char* const* argv)
{
        if (argc == 2)
{
        Tcl& tcl = Tcl::instance();
        if (strcmp(argv[1], "id") == 0)
        {
                tcl.resultf("%d", index);
        return (TCL_OK);
        }else if (strcmp(argv[1], "start") == 0)
{
        if (configured == 0)
        {
                printf("Node %d broadcasts first request message \n", index);
                WiseDAD_index = index;

                sendREQUEST_TO_JOIN();
        }

        if (configured == 1)
        {
                WiseDAD_index = index;
                sendHello();
        }

        return (TCL_OK);
}
if (strcmp(argv[1], "goodbye") == 0)
{
        printf("Node %d leaving network gracefully \n", index);
        configured = 0;
```

```
		HopCount = 0;
		ActiveNodes = AllocationTable.size();
		sendGOODBYE();
		return (TCL_OK);
}

if (strcmp(argv[1], "depart") == 0)
{
		printf("Node %d swithched off \n", index);
		configured = 0;

		ActiveNodes = AllocationTable.size();
		return (TCL_OK);
}
}

else if (argc == 3)
{
		if (strcmp(argv[1], "index") == 0)
{
		index = atoi(argv[2]);
		return (TCL_OK);
}

if (strcmp(argv[1], "set-ll") == 0)
{
		NsObject* obj;
		if ((obj=(NsObject*)TclObject::lookup(argv[2]))== 0)
{
		if (debug == 1) printf( "%s lookup of %s failed\n", argv[1], argv[2]);
		return (TCL_ERROR);
}
		ll = obj;
		return (TCL_OK);
}
}
		return Agent::command(argc, argv);
}

WiseDAD::WiseDAD(void) : Agent(PT_WiseDAD), btimer(this) , RTJ_Timer(this),
BusyTimer(this), HelloTimer(this)
{
		configured = 0;
		retries = 0;
		interval = RETRY_TIMEOUT;
		receives = 0;
		MyInitiator = 1000;
		WiseDAD_IPAddress = 0;
		AllocationTable.clear();
		HopCount = 0;
		localDAD = 0;
		packets = 0;
		localPackets = 0;
		packetNumber = 0;
		conflicts = 0;
		times = 2;
		Req = 0;
		WiseDAD_nid = 0;
		startTime = 0;
		endTime = 0;
```

```
	for (i=0; i<=99; i++)
	{
		rtable[i] = 0;
	}

	bind ("conflicts", &conflicts);
	bind ("localPackets", &localPackets);
	bind("packets", &packets);
	bind("ActiveNodes", &ActiveNodes);
	bind("WiseDAD_IPAddress", &WiseDAD_IPAddress);
	bind ("WiseDAD_nid" , &WiseDAD_nid);
	bind ("configured", &configured);
	bind("MyInitiator", &MyInitiator);
	bind("endTime", &endTime);
	bind("startTime", &startTime);
	bind("retries", &retries);
	bind("localTrials" , &localTrials);
	bind("interval", &interval);
	bind("addressSpace", &addressSpace);
	bind("debug", &debug);
	bind("state", &state); // state info at time of allocation
}

void WiseDADTimer::expire(Event*)
{

	printf(" Wise DAD timer expires \n");
	agent->dummy = 1;
	agent->sendAREP(agent->Requestor);
}

void WiseDAD_RTJTimer::expire(Event*)
{

	agent->sendREQUEST_TO_JOIN();
	printf("Node %d timer expires; Broadcasts another request packet\n", agent-
	>index);
}

void WiseDAD_BusyTimer::expire(Event*)
{
	agent->busy = 0;
}

void WiseDAD_HelloTimer::expire(Event*)
{
	agent->sendHello();
}

void WiseDAD::sendREQUEST_TO_JOIN()
{
	if (configured == 0 && index != 0)
	{

		Packet* p = Packet::alloc();
		struct hdr_cmn* ch = HDR_CMN(p);
		struct hdr_ip* ih = HDR_IP(p);
		struct hdr_WiseDAD* ah = HDR_WiseDAD(p);
		ch->size() = IP_HDR_LEN + ah->size();
		ch->addr_type() = NS_AF_NONE;
		ch->direction() = hdr_cmn::DOWN;
		ih->daddr() = IP_BROADCAST;
		ih->saddr() = this->addr();
		ah->AllocationTable.clear();
```

```
                ah->WiseDAD_type = WiseDAD_REQUEST_TO_JOIN;
                HopCount = 0;
                ah->HopCount = 0;
                // request should be rescheduled if no reply is received after a certain interval.
                RTJ_Timer.resched(interval*4);
                times = times++;
                Scheduler::instance().schedule(ll, p, 10);
}
// Here, the first node configures itself and chooses the network parameters (Network ID)
if (index == 0 && configured == 0)
{
        WiseDAD_nid = (Random::integer(addressSpace) + 2);
        configured = 1;
        WiseDAD_IPAddress = (Random::integer(addressSpace) + 2);
        printf("First Node chooses %d as its IP address and %d as the Network ID \n",
        WiseDAD_IPAddress,
        WiseDAD_nid);
        AllocationTable.push_back(WiseDAD_IPAddress);
}
}

void WiseDAD::sendCONFIRMATION(nsaddr_t dest)
{
        Packet* p = Packet::alloc();
        struct hdr_cmn* ch = HDR_CMN(p);
        struct hdr_ip* ih = HDR_IP(p);
        struct hdr_WiseDAD* ah = HDR_WiseDAD(p);
        ch->size() = IP_HDR_LEN + ah->size();
        ch->addr_type() = NS_AF_NONE;
        ch->direction() = hdr_cmn::DOWN;
        ih->daddr() = Requestor; // Chosen initiator's address
        ih->saddr() = index; // source of this message
        ah->AllocationTable.clear();
        ah->WiseDAD_type = WiseDAD_CONFIRMATION;
        ah->WiseDAD_nid = 0;
        F:\wise dad\WiseDAD.cc . File date: 2008/07/25 . File time: 03:29:56 PM
        WiseDAD.cc . Printed on 2008/08/20, 11:34:54 AM . Page 5
        ah->WiseDAD_IPAddress = 0;
        ah->HopCount = 0;
        printf("Node %d sending confirmation to new node \n", index);
        Scheduler::instance().schedule(target_, p, 0.0);
}

void WiseDAD::sendAREQ(void)
{
        Packet* p = Packet::alloc();
        struct hdr_cmn* ch = HDR_CMN(p);
        struct hdr_ip* ih = HDR_IP(p);
        struct hdr_WiseDAD* ah = HDR_WiseDAD(p);
        ch->size() = IP_HDR_LEN + ah->size();
        ih->daddr() = IP_BROADCAST;
        ih->saddr() = index;
        ah->WiseDAD_type = WiseDAD_AREQ;
        ah->WiseDAD_Request = RequestedIP;
        ah->AllocationTable.clear();
        ah->packetNumber = seqnumber;
        // increase hop count then send message
        ah->HopCount = HopCount + 1;
        // if the sender of AREQ is the source of the message, then it has to reschedule the timer
        if (HopCount == 0)
        {
                packetNumber++;
                ah->packetNumber = packetNumber;
                btimer.resched(interval*3);
```

```
                dummy = 0;
                ah->packetTime = CURRENT_TIME;
                packetTime = ah->packetTime;
                Req = ah->WiseDAD_Request;
                busy = 1;
                printf("Node %d broadcasts a AREQ packet of %d \n", index , RequestedIP);
                ah->ReversePath[0][0] = WiseDAD_IPAddress;
                ah->ReversePath[0][1] = ih->saddr();
        }

        if (HopCount > 0)
        {
                for (i=0; i<=(HopCount-1); i++)
        {
                ah->ReversePath[i][0] = ReversePath[i][0];
                ah->ReversePath[i][1] = ReversePath[i][1];
        }

        ah->ReversePath[HopCount][0] = WiseDAD_IPAddress;
        ah->ReversePath[HopCount][1] = this->addr();
        ah->packetTime = timeSent;
        }


        Scheduler::instance().schedule(ll, p, 0.0);
}

void WiseDAD::sendIPCONFLICT(nsaddr_t dest)
{
        Packet* p = Packet::alloc();
        struct hdr_cmn* ch = HDR_CMN(p);
        struct hdr_ip* ih = HDR_IP(p);
        struct hdr_WiseDAD* ah = HDR_WiseDAD(p);
        ch->size() = IP_HDR_LEN + ah->size();


        ah->HopCount = HopCount - 1;
        ih->daddr() = dest;
        ih->saddr() = index;
        ah->AllocationTable.clear();
        ah->WiseDAD_type = WiseDAD_IPCONFLICT;
        ah->WiseDAD_Request = RequestedIP;

        if (HopCount != 0)
        {
                for (i=0; i<=(HopCount-1); i++)
                {
                        ah->ReversePath[i][0] = ReversePath[i][0];
                        ah->ReversePath[i][1] = ReversePath[i][1];
                }
        printf("Node %d forwading IP Conflict to %d \n", index, ih->daddr());
        }
        Scheduler::instance().schedule(target_, p, 0.0);
}

void WiseDAD::sendAREP(nsaddr_t dest)
{
        Packet* p = Packet::alloc();
        struct hdr_cmn* ch = HDR_CMN(p);
        struct hdr_ip* ih = HDR_IP(p);
        struct hdr_WiseDAD* ah = HDR_WiseDAD(p);
        ch->size() = IP_HDR_LEN + ah->size();
        ch->addr_type() = NS_AF_NONE;
        ch->direction() = hdr_cmn::DOWN;
```

```
		Req = 0;
		ih->daddr() = Requestor;
		ih->saddr() = this->addr(); //index;
		ah->localDAD = localDAD;
		ah->DAD_trials = DAD_trials;
		ah->state = AllocationTable.size();
		ah->WiseDAD_type = WiseDAD_AREP;
		ah->WiseDAD_Request = RequestedIP;
		printf("%d sending %d to %d\n",index, ah->WiseDAD_Request, Requestor);
		ah->WiseDAD_nid = WiseDAD_nid;
		AllocationTable.push_back(RequestedIP);
		ah->AllocationTable = AllocationTable;
		ActiveNodes = AllocationTable.size();

		busy = 0;
		Scheduler::instance().schedule(target_, p, 0.0);
}

void WiseDAD::sendHello(void)
{
		Packet* p = Packet::alloc();
		struct hdr_cmn* ch = HDR_CMN(p);
		struct hdr_ip* ih = HDR_IP(p);
		struct hdr_WiseDAD* ah = HDR_WiseDAD(p);
		ch->size() = IP_HDR_LEN + ah->size();
		ch->addr_type() = NS_AF_NONE;
		ch->direction() = hdr_cmn::DOWN;
		ih->daddr() = IP_BROADCAST;
		ih->saddr() = this->addr();
		printf("Node %d sending Hello Message \n",index);
		ah->WiseDAD_type = WiseDAD_Hello;
		ah->WiseDAD_nid = WiseDAD_nid;
		HelloTimer.resched(60);

		if (configured == 0)
		{
			HelloTimer.cancel();
		}
		Scheduler::instance().schedule(ll, p, 0.0);
}
void WiseDAD::sendMerger(void)
{
		Packet* p = Packet::alloc();
		struct hdr_cmn* ch = HDR_CMN(p);
		struct hdr_ip* ih = HDR_IP(p);
		struct hdr_WiseDAD* ah = HDR_WiseDAD(p);
		ch->size() = IP_HDR_LEN + ah->size();
		ch->addr_type() = NS_AF_NONE;
		ch->direction() = hdr_cmn::DOWN;
		ih->daddr() = IP_BROADCAST;
		ih->saddr() = this->addr();
		ah->WiseDAD_nid = WiseDAD_nid;
		ah->WiseDAD_type = WiseDAD_Merger;
		if (configured == 0)
		{
			HelloTimer.cancel();
		}
		Scheduler::instance().schedule(ll, p, 0.0);
}

void WiseDAD::sendSELECT_INITIATOR(nsaddr_t dest)
{
		Packet* p = Packet::alloc();
		struct hdr_cmn* ch = HDR_CMN(p);
```

```
        struct hdr_ip* ih = HDR_IP(p);
        struct hdr_WiseDAD* ah = HDR_WiseDAD(p);
        ch->size() = IP_HDR_LEN + ah->size();
        ih->daddr() = dest;
        ih->saddr() = index;
        ah->WiseDAD_type = WiseDAD_SELECT_INITIATOR;
        ah->WiseDAD_Request = 0;
        ah->WiseDAD_nid = 0;
        printf("Node %d sending SELECT INIT to node %d \n", index, ih->daddr());
        Scheduler::instance().schedule(target_, p, 0.0);
}
void WiseDAD::sendGOODBYE(void)
{
        Packet* p = Packet::alloc();
        struct hdr_cmn* ch = HDR_CMN(p);
        struct hdr_ip* ih = HDR_IP(p);
        struct hdr_WiseDAD* ah = HDR_WiseDAD(p);
        ch->size() = IP_HDR_LEN + ah->size();
        ch->addr_type() = NS_AF_NONE;
        ch->direction() = hdr_cmn::DOWN;
        ih->daddr() = IP_BROADCAST;
        ih->saddr() = index;
        ah->WiseDAD_type = WiseDAD_GOODBYE;
        ah->WiseDAD_Request = RequestedIP;
        ah->WiseDAD_nid = WiseDAD_nid;
        ah->HopCount = HopCount + 1;
        ActiveNodes = AllocationTable.size();
        if (HopCount == 0)
        {
                ah->ReversePath[0][0] = WiseDAD_IPAddress;
                ah->ReversePath[0][1] = ih->saddr();
        }
        if (HopCount != 0)
        {
        for (i=0; i<=(HopCount-1); i++)
        {
                ah->ReversePath[i][0] = ReversePath[i][0];
                ah->ReversePath[i][1] = ReversePath[i][1];
        }
        printf("Node %d forwading Goodbye packet \n", index);
        ah->ReversePath[HopCount][0] = WiseDAD_IPAddress;
        ah->ReversePath[HopCount][1] = this->addr();
        }
        Scheduler::instance().schedule(ll, p, 0.0);
}

void WiseDAD::recv(Packet* p, Handler*)
{
struct hdr_cmn* ch = HDR_CMN(p);
struct hdr_ip* ih = HDR_IP(p);
struct hdr_WiseDAD* ah = HDR_WiseDAD(p);
nsaddr_t src = ih->saddr();
HopCount = 0;


for (i=0; i<= (ah->HopCount-1); i++)
{
        ReversePath[i][0] = ah->ReversePath[i][0];
        ReversePath[i][1] = ah->ReversePath[i][1];
}
switch(ah->WiseDAD_type) {
case WiseDAD_Hello:
if ((configured == 1) && (index > 0))
{
```

```
if (ah->WiseDAD_nid < WiseDAD_nid)
{
        printf(" Node %d detecting network merger with network %d \n", index , ah-
        >WiseDAD_nid);
        configured = 0;
        sendMerger();
        sendREQUEST_TO_JOIN();
}
}
Packet::free(p);

break;
case WiseDAD_Merger:
if (configured == 1)
{
        if (ah->WiseDAD_nid == WiseDAD_nid)
{
        printf(" Node %d receiving network merger notification \n", index);
        configured = 0;
        sendMerger();
        sendREQUEST_TO_JOIN();
}
}
Packet::free(p);
break;
case WiseDAD_REQUEST_TO_JOIN:
if (busy == 1)
{
        printf(" Node busy... %d dropping RTJ from %d \n", index , src);
}
        if (configured == 0)
{


}
if (configured != 0 && busy == 0)
{
        printf("Node %d receiving RTJ packet from %d \n", index, src);
        packets++;
        localPackets++;
        Requestor = src;
        receives++;
        busy = 1;
        BusyTimer.resched(interval*30);
        sendCONFIRMATION(src);
}
Packet::free(p);
break;
case WiseDAD_CONFIRMATION:
printf("Node %d receiving Confirmation packet from %d \n", index, src);
if (receives != 0)
{
        printf("discarding message; Already has initiator \n");
}


if (configured == 0 && receives == 0)
{
        startTime = (Scheduler::instance()).clock();
        packets++;
        localPackets++;
        MyInitiator = src;
        receives++;

        RTJ_Timer.cancel();
```

```
            sendSELECT_INITIATOR(src);
}
Packet::free(p);
break;
case WiseDAD_AREP:
printf("Node %d receiving AREP packet from %d with IP address of %d \n", index, src, ah-
>WiseDAD_Request);
// no need for this since AREP is a unicast message
if (configured == 0 && MyInitiator != src)
{
            printf("Message discarded, not mine");
}
if (configured == 0 && MyInitiator == src)
{
            // a new node configures itself after a successful WiseDAD
            packets++;
            localPackets++;
            WiseDAD_IPAddress = ah->WiseDAD_Request;
            WiseDAD_nid = ah->WiseDAD_nid;
            AllocationTable = ah->AllocationTable;
            configured = 1;
            endTime = (Scheduler::instance()).clock() ;
            state = ah->state;
            retries = ah->DAD_trials;
            localTrials = ah->localDAD;

            printf(" Node %d now configured with %d as its IP address and %d as NetID \n",
            index,

            WiseDAD_IPAddress,WiseDAD_nid);

            ActiveNodes = AllocationTable.size();

            sendHello();
}
Packet::free(p);
break;
case WiseDAD_IPCONFLICT:
if (configured ==1)
{
            HopCount = ah->HopCount;
            RequestedIP = ah->WiseDAD_Request;
            if (ah->HopCount > 0)
{

            sendIPCONFLICT(ah->ReversePath[ah->HopCount-1][1]);

}
if (ah->HopCount == 0)
{
            printf("Node %d receiving IP Conflict packet from %d \n", index, src);

            packets++;
            localPackets++;

            found = 1;
            ActiveNodes = AllocationTable.size();
do
{

            RequestedIP = (Random::integer(addressSpace) + 2);
            found = 0;
            localDAD++;
```

```
            for (i=0; i<= (ActiveNodes-1); i++)
            {
            if (AllocationTable[i] == RequestedIP)
            {
            found = 1;
            }
            }
}
while ((found == 1));
HopCount = 0;

DAD_trials = DAD_trials + 1;
sendAREQ();
}
ActiveNodes = AllocationTable.size();
} Packet::free(p);
break;
case WiseDAD_AREQ:

if (configured == 0)
{

}
old = 0;
if (index == ah->ReversePath[0][1])
{
            old = 1;
}
if ((ah->packetNumber <= rtable[ah->ReversePath[0][1]]) && (old == 0))
{
            old = 1;
            rtable[ah->ReversePath[0][1]] = ah->packetNumber;

}

if ((Req == ah->WiseDAD_Request) && (old == 0) && (configured == 1))
{
            printf("Node %d receiving AREQ packet from %d \n", index, src);
            rtable[ah->ReversePath[0][1]] = ah->packetNumber;
            sendIPCONFLICT(ah->ReversePath[ah->HopCount-1][1]);
            old = 1;
}

if (configured == 1 && old == 0)
{
            printf("Node %d receiving AREQ packet from %d \n", index, src);
            rtable[ah->ReversePath[0][1]] = ah->packetNumber;
            seqnumber = ah->packetNumber;
            packets++;
            HopCount = ah->HopCount;
            RequestedIP = ah-> WiseDAD_Request;
            timeSent = ah->packetTime;
            found = 0;
            i = 0;
            //updating allocation table
for (i = 0; i<= (ah->HopCount-1); i++)
{
            AllocationTable.push_back(ah->ReversePath[i][0]);
}

ActiveNodes = AllocationTable.size();
for (int k=0; k<=(ActiveNodes-2); k++)
{
            for (i=(k+1); i<=(ActiveNodes-1); i++)
            {
```

```
                        if (AllocationTable[k] == AllocationTable[i])
                {

                AllocationTable.erase(AllocationTable.begin() + k);
                ActiveNodes = AllocationTable.size();
                k--;
                i--;
}
}
}
ActiveNodes = AllocationTable.size();
if (WiseDAD_IPAddress != ah->WiseDAD_Request)
{
                printf("Node %d forwarding AREQ \n", index);
                HopCount = ah->HopCount;
                sendAREQ();
}
if ((WiseDAD_IPAddress == ah->WiseDAD_Request))
{
                conflicts++;
                sendIPCONFLICT(ah->ReversePath[ah->HopCount-1][1]);
                printf("Sending conflict message: Defending my IP of %d \n"
                ,WiseDAD_IPAddress);
}
ActiveNodes = AllocationTable.size();
}

Packet::free(p);
break;
case WiseDAD_SELECT_INITIATOR:
//try and check if message is mine
// there is need to check allocation table first before AREQ (Local DAD)
printf("Node %d receiving SELECT INIT packet from %d \n", index, src);
counter = 0;
found = 1;
HopCount = 0;
BusyTimer.cancel();
busy = 1;
localDAD = 1;
ActiveNodes = AllocationTable.size();
packets++;
localPackets++;

do
{
                counter = counter +1;

                RequestedIP = Random::integer(addressSpace) + 2;
                printf("Node %d generating %d for new node \n", index , RequestedIP);
                found = 0;


for (i=0; i<= (ActiveNodes-1); i++)
{
                if (AllocationTable[i] == RequestedIP)
{
found = 1;
localDAD++;
}
}
} while ((found == 1));
HopCount = 0;
DAD_trials = 1;
sendAREQ();
```

```
Packet::free(p);
break;
case WiseDAD_GOODBYE:
//checking if this node has received this message before
// and discarding the message
printf("Node %d receiving Goodbye packet from %d \n", index, src);
if (configured == 0)
{
        printf(" Discarding the message, not configured \n");
}
if (configured == 1)
{
        packets++;
        HopCount = ah->HopCount;
        RequestedIP = ah->WiseDAD_Request;
        found = 0;
        i = 0;
        printf("Hop count is %d \n",ah->HopCount);
        //check if message is new or old
for (i=0; i<=(ah->HopCount-1); i++)
{
        if (ah->ReversePath[i][0] == WiseDAD_IPAddress)
{
        found = 1;
}
}
if (found == 1)
{
        printf("Old message: discarding it \n");
}
// if message is new then process it.
if (found == 0)
{
//updating allocation table
for (i = 0; i<= (ah->HopCount-1); i++)
{
        AllocationTable.push_back(ah->ReversePath[i][0]);
}
ActiveNodes = AllocationTable.size();
for (int k=0; k<=(ActiveNodes-2); k++)
{
for (i=(k+1); i<=(ActiveNodes-1); i++)
{
        if (AllocationTable[k] == AllocationTable[i])
        {
        AllocationTable.erase(AllocationTable.begin() + k);
        ActiveNodes = AllocationTable.size();
        k--;
        i--;
        }
}
}
sendGOODBYE();
}
ActiveNodes = AllocationTable.size();
} Packet::free(p);
break;}}
```

APPENDIX C: Wise-DAD TcL Script

```
#===================================================================
# Define options
#===================================================================
set val(chan) Channel/WirelessChannel ;          # Channel type
set val(prop) Propagation/TwoRayGround ;         # radio-propagation model
set val(ant) Antenna/OmniAntenna ;               # Antenna type
set val(ll) LL ;                                 # Link layer type
set val(ifq) Queue/DropTail/PriQueue ;           # Interface queue type
set val(ifqlen) 50 ;                             # Max packet in ifq
set val(netif) Phy/WirelessPhy ;                 # Network interface type
set val(mac) Mac/802_11 ;                        # Mac type
set val(rp) DSR ;                                # ad-hoc routing protocol
set val(nn) 100 ;                                # number of mobile nodes
set val(x) 1000
set val(y) 1000
set val(seed) 0.0
set val(sc) scen-50-500x500
set val(stop) 6000.0 ;                           # simulation time
set val(god) off
set val(intv) 0.5 ;                              # broadcast interval (in seconds)
set val(limit) 254
set ns_ [new Simulator]
$ns_ use-scheduler Heap
set tracefd [open WiseDAD.tr w]
$ns_ trace-all $tracefd
set namtrace [open WiseDAD-out.nam w] ;          # for wireless traces
$ns_ namtrace-all-wireless $namtrace $val(x) $val(y)

set f0 [open WiseDAD-all.tr w]
set f1 [open WiseDAD-com.tr w]
set f2 [open WiseDAD-lat.tr w]



# for all trace
puts $f0 "ID IPaddress latency localPackets globalPackets totalPackets"
# communication overhead
puts $f1 "ID localPackets globalPackets totalPackets DADTrials LocalRetries state_at_alloc
AllocationTable"
# for latency
puts $f2 "ID IPaddress startTime endTime latency DADTrials initiator"
proc finish {}
{
        global f0 f1 f2
        close $f0
        close $f1
        close $f2
        #exec nam WiseDAD-out.nam
        exit 0
}

proc record { WiseDAD }
{
        global f0 f1 f2
        set index [$WiseDAD id]
        set DADretrs [$WiseDAD set retries]
        set LocalRetries [$WiseDAD set localTrials]
        set address [$WiseDAD set WiseDAD_IPAddress]
        set startProcess [$WiseDAD set startTime]
        set endProcess [$WiseDAD set endTime]
```

```
        set initiator [$WiseDAD set MyInitiator]
        set dataReceived [$WiseDAD set packets]
        set localData [$WiseDAD set localPackets]
        set state_at_alloc [$WiseDAD set state]
        set AllocationTable [$WiseDAD set ActiveNodes]
        set conflictMessages [$WiseDAD set conflicts]
        set globalData [expr $dataReceived - $localData]
        set latency [expr $endProcess - $startProcess]
        # for all trace
        puts $f0 "$index $address $latency $localData $globalData $dataReceived"
        # communication overhead
        puts $f1 "$index $localData $globalData $dataReceived $DADretrs $LocalRetries
        $state_at_alloc $AllocationTable"
        # for latency
        puts $f2 "$index $address $startProcess $endProcess $latency $DADretrs $initiator"
}

#
# Define topology
#
set topo [ new Topography]
$topo load_flatgrid $val(x) $val(y)
#
# Create GoD
#
set god_ [create-god $val(nn)]
set chan_1_ [new $val(chan)]
#
#Configure nodes
#
$ns_ node-config -llType $val(ll) \
-adhocRouting $val(rp) \
-macType $val(mac) \
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
-topoInstance $topo \
-channel $chan_1_ \
-agentTrace ON \
-routerTrace ON \
-macTrace OFF \
-movementTrace OFF

for { set i 0 } { $i < $val(nn) } { incr i }
 {
        set node_($i) [$ns_ node $i]
        $node_($i) random-motion 0 ; # disable random motion
        $god_ new_node $node_($i)
}

set maxX 1200
set maxY 1200
for { set i 0 } { $i < $val(nn) } { incr i }
{
        $node_($i) set X_ [expr {int(rand()*$val(x))}]
        $node_($i) set Y_ [expr {int(rand()*$val(y))}]
        $node_($i) set Z_ = 0.0
}
#
# Define initial position in nam
#
for { set i 0 } { $i < $val(nn) } { incr i }
```

```
{
        $ns_ initial_node_pos $node_($i) 50
}
# Setup a the WiseDAD agent
for { set i 0 } { $i < $val(nn) } { incr i }
{
        set WiseDAD_($i) [new Agent/WiseDAD $i]
        $WiseDAD_($i) index $i
        $WiseDAD_($i) set interval $val(intv)
        $WiseDAD_($i) set addressSpace $val(limit)
        $WiseDAD_($i) set debug 1
        $ns_ attach-agent $node_($i) $WiseDAD_($i)
        set ll($i) [$node_($i) set ll_(0)]
        $WiseDAD_($i) set-ll $ll($i)
}

for { set i 0 } { $i < $val(nn) } { incr i }
{
        $ns_ at [expr $i*30] "$WiseDAD_($i) start"
}

#
# Tell nodes when the simulation ends
#

for {set i 0} {$i < $val(nn)} {incr i}
{
        $ns_ at $val(stop) "record $WiseDAD_($i); $node_($i) reset"
}

$ns_ at $val(stop).0002 "puts \"NS EXITING...\"; $ns_ halt; finish"
puts "Starting simulation..."
$ns_ run
```