IP Address Auto-Configuration for Wireless Ad-hoc Networks

MURIMO BETHEL MUTANGA

200711454

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

Doctor of Philosophy (Computer Science)

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

2019

Declaration

I declare that this dissertation is my own original work, conducted under the supervision of Professor Matthew O. Adigun. It is submitted for the degree of Doctor of Philosophy (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. Some parts of this thesis have been published in conference proceedings and journals. Some of these papers are contained verbatim in this thesis. All other sources used in the dissertation have been duly acknowledged.

Signature: _____

MUTANGA MB

Dedication

This piece of work is dedicated to the Mutanga empire, the empire that made me

Acknowledgements

I would like to express my appreciation to Prof M.O. Adigun for providing dedicated supervision throughout my research. Financial support by Telkom, Technology and Human Resources for Industry Programme (THRIP) and Huawei to the Department of Computer Science at the University of Zululand through the Centre of Excellence (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. 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:

- Paul Tarwireyi his constructive criticisms; Without his contributions, this work would not have seen the light of day; His guidance helped me in all the time of research and writing of this thesis.
- My wife, Tarirai, for the emotional support and encouragement.
- My student, Dominic for helping with editing and referencing within the thesis.
- My mother, MaNcube, for encouragement and support every step of way.

I would also like to thank daughters for bearing with me for my long absence from home during the final stages of the thesis.

My sincere gratitude also goes to my late grandmother, MaSibanda for her prayers. Special mention goes to the Mutanga empire and my late father whose immense contribution to my life cannot be expressed in words.

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.

List of Publications

Mutanga, M.B., Mudali, P., Chani, T., Mhlanga, M. and Adigun, M.O., 2011. The effect of DAD timeout period on address autoconfiguration in wireless ad-hoc networks. SATNAC.

Mutanga, M.B., Mudali, P. and Adigun, M.O., 2011. Towards auto-configuring routing protocols for wireless ad-hoc networks. International Journal of Computer Engineering Research, 2(2), pp.19-27.

Mutanga, M.B., Tarwireyi, P. and Adigun, M., 2015, November. Handling network merging and partitioning in MANETs. In New Technologies of Information and Communication (NTIC), 2015 First International Conference on (pp. 1-6). IEEE.

Mutanga, M.B., Adigun, M. and Chani, T., 2015, September. The effect of network traffic on Duplicate Address Detection in wireless ad-hoc networks. In Computer Networks and Information Security (WSCNIS), 2015 World Symposium on (pp. 1-6). IEEE.

Table of Contents

List of Figures and Tables

Abstract

In an ad hoc network, nodes collaborate to allow communication without the presence of network infrastructure. Lack of manual management in ad hoc networks means that automatic configuration is highly desirable. The need for automatic configuration capabilities will become significantly more intense 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. High levels self-organisation provides an out-of-the-box functionality such that very little technical expertise is required to set up a network. However, efficiently providing unique IP addresses in ad hoc networks is still an open research question. This study is a successful attempt to investigate automatic IP addressing in wireless ad hoc networks as both Multicriteria decision making (MCDM) problem and a challenge of building a system that converges towards the global desired goal. Consequently, the solution proposed in this thesis is inspired by observing swam systems' ability to converge towards a global goal from local interactions. The investigation reported in this thesis first answered the question of how different network conditions affect the address auto-configuration process. Experiments to investigate the effect of mobility, network traffic and DAD timeout period on address auto-configuration were conducted. The results of these experiments informed the design of the new protocol, D-DAD, proposed in this work. The proposed IP address auto-configuration mechanism was simulated in Ns2 and results were compared with existing Wise-DAD and StrongDAD protocols. We performed five experiments to investigate the effect of network size, node density, node arrival rate, mobility, and network traffic on communication overhead, address uniqueness and latency. The results showed that D-DAD outperformed StrongDAD in all the metrics used for comparison. However, in some instances, D-DAD recorded more communication overhead in comparison to Wise-DAD but had better latency and fewer address conflicts.

Chapter 1

Introduction

1.0 Preamble

Wireless ad hoc networks dynamically self-configure and organise themselves, (with the node establishing connectivity automatically), and maintain mesh connectivity among themselves. Distributed algorithms that control wireless ad hoc networks make them resilient to faults and make it easy to establish and maintain a network with no need for infrastructure. These and many other features bring many advantages such as low up-front cost, easy network maintenance, robustness, and reliable service coverage.

Over the years, wireless technologies have evolved rapidly to provide better services with lower costs in deployment and administration (Bernardos et al., 2005; Hossain et al., 2014). In particular, wireless ad hoc networking has proved to be a promising technology in many application scenarios (Sharma, 2013). These include tactical military deployments, low cost internet connectivity, and disaster recovery operations. The definition of wireless ad hoc networks has expanded to include Mobile Ad Hoc Networks (MANETs), Wireless Sensor Networks (WSNs) Wireless Mesh Networks (WMNs), Smart Phone Ad Hoc Networks (SPANs), and Vehicular Ad Hoc Networks (VANETs) (Loo et al., 2016). Application scenarios of these networks require a high degree of self-organisation and configuration.

Due to the requirement that nodes in wireless ad hoc networks must dynamically self-configure, algorithms to handle different aspect of self-configuration and self-organisation are of paramount importance. One important network parameter that requires dynamic configuration is an IP address. This is highly desirable because nodes require IP addresses to send and receive both unicast and multicast messages. Most importantly, automatic configuration of nodes reduces network administration and makes it possible to construct a network on the fly.

In essence, automatic configuration provides an out-of-the-box functionality such that very little technical expertise is required to set up and run a computer network. This is of fundamental significance in rural areas where technical expertise is not readily available. Even if the technical expertise could be found, manually configuring potentially hundreds of devices would be too time-consuming and error- prone. Application scenarios such as disaster recovery also require that networks be set up with minimum human configuration with no physical infrastructure.

The need for automatic configuration capabilities becomes significantly more intense 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 (Bernardos et al., 2005; Mennicken et al., 2014). Many algorithms for automatic configuration of IP addresses have been proposed in the literature.

In wireless ad hoc networks, IP address auto-configuration protocols are classified as being either stateful or stateless. Stateful approaches try to mimic the Dynamic Host

Configuration Protocol (DHCP) server system that is used in wired networks. The addresses that are free are known in advance and are maintained by one or more nodes in the network. New nodes are configured from the known pool of IP addresses. On the other hand, stateless protocols follow the trial and error method. Only the range of allowed IP addresses is known. A new node generates an IP address that is within the permitted range and checks for uniqueness through a network -wide broadcast called Duplicate Address Detection (DAD) procedure. Many variations of the two approaches have been proposed and presented in the literature. Another school of thought that combines both stateless and stateful approaches has also been a subject of debate for some time. This paradigm is known as hybrid auto-configuration.

The main challenge of the stateful paradigm is reliable synchronisation of address allocation tables, given the unpredictable environments that characterise wireless ad hoc networks (Suganthi & Ravimaran, 2014). Inconsistent address allocation tables results in the configuration of duplicate addresses. Explicit synchronisation of state information is a major challenge and is associated with high communication and processing overhead. However, explicit state synchronisation is also associated with low address conflicts.

In stateless approaches the most important aspect is the design of the Duplicate Address Detection (DAD) mechanism. A poorly designed DAD mechanism may result in address conflicts if nodes cannot defend their IP addresses. However, according to Grajzer & Głąbowski (2016) performing a DAD procedure is the best way of making sure that an IP address is unique. This thesis adopted Grajzer and Glabowski's (2016) philosophy on the

importance of having a Duplicate Address Detection mechanism. However, the design of DAD is not a trivial task. The best values for DAD timeout and the number of DAD trials have not been established in the literature.

This thesis argues that the current approaches fail to address the auto-configuration problems due to the challenges posed by the unpredictable nature of wireless ad hoc networks. Nodes in a wireless ad hoc network environment may be mobile, thereby affecting the delivery of critical IP address configuration messages. Connections in wireless ad hoc networks are usually not reliable and are at times unidirectional. How this affects the auto-configuration, has not, to the best of our knowledge, been investigated. Basically, proposals in the literature have not investigated how different network conditions that characterise a wireless ad hoc network environment affect the functioning of the solution.

Furthermore, this thesis is of the opinion that the design of an IP address auto-configuration should, among other things, be informed by how network conditions affect the process of address auto-configuration. This allows for the auto-configuration components and other auto-configuration parameters to adapt to changes in the network environment. Awareness of the effect of network conditions will aid in effective management of the IP address space. As a result, the solution proposed in this thesis is inspired by swam systems where local interactions lead to the desire global behavior that emerges over time.

1.2 Statement of the Problem

Debate around how to effectively allocate IP addresses in wireless ad hoc networks has been around for a very long time. Three dimensions to the solutions have been proposed, namely, stateless, stateful and hybrid approaches. A number of solutions following these paradigms have been developed and tested.

The primary reason why there has been so much work done with no clear solution is because of the unpredictable nature of wireless ad hoc networks. Unpredictable wireless links, mobility, and ever-changing membership are some of the conditions that make it difficult to arrive at a solution. In address auto-configuration, for example, the goal of the auto-configuration protocol is to configure nodes with low or no address conflicts in a reasonable amount of time and with low communication overhead. All this should be achieved in the context of a highly dynamic environment.

Despite the number of protocols reported in the literature researchers have not investigated how network conditions such as mobility, unreliable links, and network traffic affect the IP address automatic configuration process.

It is clear that developing solutions for wireless ad hoc networks lies in understanding how different network conditions affect the functioning of the protocols. This understanding can then be used to develop new protocols that are more robust and adaptive to network conditions.

1.3 Research Questions

In order to tackle the identified research issues, the following research questions were formulated:

(a) What is the appropriate IP address auto-configuration paradigm that can handle network dynamics?

- This thesis investigated why existing paradigms do not handle network dynamics well.

(b) Why are address auto-configuration algorithms not resilient to different network conditions?

- The following investigations were conducted to answer this question:
  - Determining the effect of network merging on address auto configuration;
  - Determining the effect of network partitioning on address auto-configuration; and
  - Determining the effect of network traffic and mobility on address auto- configuration.

(c) What are the best configurations for DAD that will result in low address conflicts and low communication overhead?

- The following investigations were carried out to answer this question:
  - Determining the best value of DAD timeout; and
  - Determining the optimal number of DAD trials.

(d) How can the wireless ad hoc networking environment inform the design of address auto-configuration protocols that can adapt to different network conditions?

- This part of the research explored how the results from the first three questions can be used in the design of Address Auto-configuration protocols. The Dynamic Duplicate Address Detection (D-DAD) address auto-configuration was formulated and evaluated.

1.4 Rationale of the Research

Wireless communications have become a de facto means of connectivity in today's world. The growing deployment rate of wireless networks is evidence that wireless networking is rapidly becoming a predominant means of electronic communication (Chin et al., 2014). Wireless technologies are used to interconnect a vast number of devices without the requirement for cables and with minimal network infrastructure.

Wireless ad hoc networks, in particular, have the potential of connecting different devices ranging from computers to sensors (Xu et al., 2014). Application scenarios such as disaster recovery, search and rescue, and military networks all benefit from the self organising characteristics of wireless ad hoc networks. Due to the potentially large network sizes of wireless ad hoc networks, automatic configuration of IP and other network parameters is important. Manually configuring potentially hundreds of devices would be too time-consuming and prone to human error. Automatic configuration minimises the requirement for manual configuration, thereby bringing an out-of-the-box functionality such that very

little technical expertise is required to set up a network. This is of paramount importance in application scenarios such as disaster recovery, and search and rescue. 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. Connecting the entire home to the internet with endless workability and entertainment possibilities, as well as the ability to manage security and passwords from one home system. Users can store files on one file storage app and access it from anywhere in the home environment with any device (Nag et. al, 2017). In addition, automatic configuration makes ad-hoc networks suitable for many other applications like military, disaster area, rescue operation, collaborative computing and conference meeting where it is not possible to setup wired network or infrastructure based wireless network.

1.5 Goal

The goal of this thesis was to develop IP address auto-configuration algorithms for wireless ad hoc networks.

1.6 Objectives

To address the research questions formulated in this thesis the following objectives were set:

i. Investigate the best values of DAD timeout and DAD trials during address auto-configuration.
ii. Investigate the effect of network traffic and mobility on address auto-configuration.
iii. Design and evaluate appropriate network merging and partitioning mechanisms.
iv. Propose an address auto-configuration protocol.
v. Simulate and evaluate the proposed address auto- configuration protocol.

1.7 Methodology

The following research methodology was followed in this thesis:

a) Literature Survey

A literature survey was conducted to get an in-depth understanding of what other scholars have done to address the identified problems. This part of the research was based mainly on critical evaluative and comparative analysis of existing related works by other scholars.

The survey of literature resulted in the model proposed in chapter 2 of this thesis. The solution developed to address the identified problem is based on this model.

b) Design Science

The goal of this research was to develop an address auto-configuration that takes into account network conditions. The Design Science Research Methodology (DSRM) was adopted because it is important for conducting research in disciplines oriented to creating artifacts that serve as solutions to defined problems. Design science research focuses on the development and performance of artifacts with the explicit intention of making functional improvements of the artifact (Yin, 2017). The artifacts created in the design science research process include algorithms, computer interfaces , and system design methodologies or languages(Gregor and Hevner, 2013). Since the goal of this work included making improvements on existing IP address auto-configuration protocols, design science was found to be the most appropriate method. We adopted the steps outlined in (Peffers et al., 2008), namely:

i. Problem Definition and Motivation

This step took input from the statement of the problem and used the information gathered from the literature survey to motivate the relevance of the study. The survey was also used as a tool to acquire useful knowledge for developing the solution approach.

ii. Investigation by Literature Survey

The investigation was in two phases, the first being a literature survey to establish the state of the art in the field of IP address auto-configuration. After identifying issues that needed investigation, investigative experiments were carried out to validate the findings. The feedback obtained from the investigative experiments reported in chapters 3, 4 and 5 was then used to formulate the solution presented in chapter 6 and tested in chapter 7.

iii. Design and Development

Applying self-organisation to communications networks requires a constructive engineering approach, hence the feedback from the literature survey was used to formulate the processes, and methods or algorithms that contributed towards the development of the model proposed in chapter 2. Design of the algorithms presented in this thesis was based on the investigations conducted in chapters 3, 4, and 5.

iv. Demonstration

For the purpose of proof of concept, simulation experiments in an NS2 simulator were conducted. NS2 is an open source, event driven simulation tool that has become the de facto standard in the simulation of wireless ad hoc networks. Using NS2 allowed for the simulation of large network sizes and vary different network conditions with ease.

v. Evaluation

A performance analysis of the proposed model was done using both graphical and theoretical techniques. The IP address auto-configuration proposed in this work was evaluated using the following metrics:

a. Latency
b. Address conflicts
c. Communication overhead

In the experiments conducted the following network conditions were varied:

i. Network traffic
ii. Node arrival rate
iii. Node density
iv. Mobility
v. Network size

1.8 Synopsis of the Thesis Contributions

This thesis presents a successful investigation of the problem of automatic address allocation in wireless ad hoc networks. Below is the summary of the contributions of this thesis:

(a) This thesis proposed algorithms that take network conditions into account when performing address allocation. The thesis took a paradigm shift by advocating for adaptation whereby parameters such as DAD timeout period are determined at runtime.

(b) Furthermore, this thesis has advanced the design of the DAD mechanism by establishing the optimal values for the DAD timeout period. Current DAD-based protocols in the literature set DAD timeout at 1.8 seconds. This work, however concluded that DAD timeout should vary depending on network conditions.

(c) This thesis has shaped the solution space of address allocation protocols by investigating the effect of network traffic and mobility of address allocation protocols. The results obtained in the investigations compel researchers to look at address allocation solutions differently. Current solutions proposed in the literature do not consider this important aspect.

(d) Network partitioning algorithms proposed in this thesis can distinguish between temporary and permanent partitions. Being able to distinguish between temporary and permanent partitions removes the burden of unnecessary address and network ID changes which can cause a lot of communication overhead.

(e) As an improvement to network merging solutions proposed in the literature, this thesis proposed an algorithm that handles the merging of networks that were previously part of the same network without changes in IP addresses.

(f) This thesis established a relationship between node density and the performance of an address auto-configuration protocol. These findings are important for planning node placement for network deployment.

1.9 Thesis Structure and Composition

The rest of the thesis is organised as follows:

Chapter 2: The IP address Space Management Problem in Wireless Ad hoc Networks: Basic background concepts concerning the area of IP address auto-configuration are introduced. Different approaches to automatic configuration are reviewed in this chapter. The chapter concludes by giving a solid direction to the design of IP address auto-configuration protocols. A new paradigm of allocating IP addresses is proposed.

Chapter 3: Determining the optimal DAD configuration parameters: Experiments to test the traditional DAD procedure were conducted in this chapter. This was motivated by the conclusion drawn in chapter 2 establishing the need to make more investigations on how the process of DAD is done. Results reported in this chapter were used to design the address auto-configuration protocol described in chapter 5.

Chapter 4: Effect of Network Traffic and Mobility on Address Auto-configuration: Evidence in the literature shows that network conditions such as mobility, traffic type and volume affect the functioning of network layer protocols. However, to the best of our knowledge, no investigation has been conducted to determine how different network conditions affect address auto-configuration protocols. This chapter, therefore, investigates how network traffic affects the performance and functions of address auto-configuration protocols. Knowing how network conditions affect address auto-configuration helps in designing better protocols.

Chapter 5: IP Address Auto-configuration Algorithms for Wireless Ad hoc networks: In this chapter an address auto-configuration protocol (D-DAD) based on the investigations done in chapters 2, 3, and 4, is designed.

Chapter 6: Performance evaluation of the D-DAD address allocation algorithms: In this chapter the address allocation mechanism proposed in chapter 5 is evaluated. Comparison with previous works is conducted.

Chapter 7: Thesis Contributions, Conclusion and Future Work: This chapter presents the contributions made in this thesis. A summary of the results obtained in earlier chapters is used in clarifying the contributions. Future direction, open issues, limitations and conclusions are also given in the chapter.

Chapter 2

The IP address Space Management Problem in Wireless Ad hoc Networks

2.0 Introduction

The wireless ad hoc network environment presents researchers with challenges when designing protocols at different layers. IP address auto-configuration solutions, in particular, have the challenge of coping with a highly dynamic environment and uncertain network conditions (Fan & Subramani, 2005; Levin et a.l, 2014). The previous chapter identified the need to make thorough investigation of how the ad hoc networking environment affects the functioning of address auto-configuration protocols. The purpose of having an address auto-configuration protocol is to manage the IP address space and configure nodes with unique addresses. However, IP addresses come from a finite domain, hence dealing with the management of a limited resource in an uncertain environment is not an easy task.

This chapter envisages a paradigm shift in the problem of address auto-configuration. The paradigm presented in this chapter advocates for address auto-configuration mechanisms to adapt to network conditions such as mobility and high traffic volume. This contribution answered the second research question which aimed at investigating the best approach to the auto-configuring problem. An analysis of different approaches to the management of

the IP address space reported in this chapter concluded that due to the rigidity of current approaches, achieving flawless auto-configuration is a challenge, hence the new approach.

The idea presented in this chapter has one distinct advantage over existing schools of thought in that it takes into account changes in network conditions. Network conditions such as mobility, high traffic volume, network merging and partitioning affect the functioning of address auto-configuration mechanisms and hence must be taken into account when designing auto-configuration solutions. With the adaptive model, nodes will have to align with the settings of an auto-configuration protocol to suit the current network conditions.

The rest of this chapter is organised as follows: In section 2.1 a discussion on some design issues in IP address auto-configuration is presented. The discussion presented in section 2.1 lays a foundation for analysing the best building components of an address auto-configuration mechanism. In Section 2.2 related approaches in the area of IP address auto-configuration are described. Section 2.3 describes the proposed IP address space management model. Section 2.4 concludes the chapter.

2.1 Design Issues in IP address Auto-configuration

The wireless ad hoc network environment presents researchers with unique challenges when designing IP address auto-configuration protocols. In view of the wide array of constraints, a protocol for assigning IP addresses in wireless multi-hop networks should meet the following requirements:

a) Interoperability

IP address auto-configuration solutions should allow for interoperability with traditional IP based networks (Kim et al., 2015). Any solution aimed at working in this kind of environment must be compatible with standard nodes, otherwise no major changes to the protocol stack that may affect interoperability can be made.

b) Merging support

This characteristic basically deals with the ability of an auto-configuration mechanism to detect network merging. When network merging is detected the protocol should invoke the functionalities that eliminate IP address conflicts and connectivity problems (Abid et al., 2015). In wireless ad hoc networks merging can take place when at least two independently configured networks come into each other's transmission range, thereby forming one network. It is therefore imperative that an IP auto-configuration solution aimed at supporting wireless ad hoc networks should provide support for handling network merging. In essence, the solution must not disrupt the functioning of the merging networks. (A solution to the network merging problem is presented in chapter 5).

c) Partitioning support

An IP auto-configuration mechanism must have the ability to detect network partitioning. In a typical wireless ad hoc network, nodes can randomly be switched off and the network may be partitioned. Such an occurrence may be temporary or permanent. Therefore, an IP auto-configuration solution aimed at working in wireless ad hoc networks is expected to take these two scenarios into consideration. If the disconnection is temporary the solution should be able to handle the merging of the networks at a later time without much disruption to the network's performance. On the other hand, if the disconnection is permanent, the remaining network must be able to re-use the IP addresses on the disconnected segment. Contrary to other solutions in the literature, the one proposed later in chapter 5 addresses this issue of being able to distinguish between temporary and permanent network partitioning.

d) Robustness and Fault tolerance

Due to the unpredictable nature of the wireless ad-hoc networking environment, protocols ought to be fault tolerant and robust. Given the multi-hop characteristic of ad hoc scenarios, it is important to analyse the design assumptions underlying an IP auto-configuration mechanism. IP auto-configuration mechanisms aimed at working in wireless ad hoc networks should be robust in terms of resiliency to sporadic transmission problems, mobility, and packet loss. To increase robustness and fault tolerance, the paradigm shift proposed in this chapter advocates the monitoring of network conditions that affect the auto-configuration process.

e) Latency

Another important characteristic, related to robustness, is the latency of an auto-configuration solution. Depending on the scenario and/ or the application, latency can be defined as the time required by a single node to get a usable, unique and valid IP address. It is important that the auto-configuration protocol configure nodes with low latency.

f) Security

Wireless ad hoc networks have unique characteristics, thereby making it difficult to address security and authentication issues. The work by Kumar et al., (2008) and Rehman & Manickam (2015), enumerated possible attacks to the IP auto-configuration process. These attacks include Address Spoofing, Address Conflict, Address Exhaustion, and Negative Reply. Most protocols do not address security during auto-configuration at all. For example, proposals in Güne & Reibel (2002), Fazio et al., (2006), Indrasinghe, Indrasinghe et al., (2006), Kim et al., (2007), Mutanga et al., (2008) and Ramakrishnaiah & Reddy (2016) only addressed the auto-configuration problem whilst the security issues surrounding this aspect were not addressed. Pan et al., (2005), Zakaria et al., (2015) and Praptodiyono et al., (2015) are some of the few proposals that consider security during automatic configuration. Their proposal binds each IP address with a public key, allows a

node to self-authenticate itself, and thus thwarts address spoofing and other attacks associated with auto-configuration.

g) Scalability

In most cases the process of IP address auto-configuration requires that nodes exchange a number of messages before a node can be allocated an IP address. These messages might either be flooded in the network or exchanged locally, and they usually grow with network size, leading to high overhead (Harish et al., 2008), (Pathan, 2016). Stateless approaches degrade dismally when the network grows because of the flooding mechanism that is used to detect duplicate IP addresses. Both communication overhead and latency are generally high in this approach. Some stateful approaches, e.g. Prophet (Zhou et al., 2013), tried to address this problem by configuring nodes using local messages only. This, however, compromises on the uniqueness of the address. The biggest challenge in building scalable protocols, therefore, is to try and reduce communication overhead without compromising on address uniqueness and latency. The range of IP addresses should also be scalable. IP addresses should not run out of availability when a large number of nodes are joining (Harish et al., 2008). This thesis is of the view that scalability can be enhanced by avoiding network-wide broadcasting of messages and elimination of explicit state synchronisation.

h) Duplicate Address Detection

Duplicate address detection is required when two independent networks merge or as a continuous process to resolve duplicate IP addresses that might arise as a result of

erroneous address allocation. To provide this capability to an auto-configuration protocol there need to determine how duplicate addresses are detected and how address conflicts will be resolved. In this subsection we give an analysis of the problem of address duplication. We modelled the problem using the birthday problem. In probability theory, the birthday problem or birthday paradox concerns the probability that, in a set of n randomly chosen people, some pair of them will have the same birthday. By the pigeonhole principle, the probability reaches 100% when the number of people reaches 366. However, 99.9% probability is reached with just 70 people, and 50% probability with 23 people. The birthday paradox has been used in estimating a lot of problems including transitivity in knowledge management (Jha et al 2015).

In the case IP addressing, the goal is to compute P(A), the probability that at least two nodes are the same ie two nodes have duplicate IP addresses. However, it is simpler to calculate P(A′), the probability that no at least two elements that are the same. Then, because A and A′ are the only two possibilities and are also mutually exclusive, therefore we can calculate the probability as follows: $P(A) = 1 - P(A')$.

It is easier to first calculate the probability p(n) that all n nodes have unique IP addresses. According to the pigeonhole principle: for natural numbers k and m, if $n = km + 1$ objects are distributed among m sets, then the pigeonhole principle asserts that at least one of the sets will contain at least $k + 1$ objects. This means that p(n) is zero when the number of nodes is more than the number of IP addresses allowed in the network, say L. When $n \leq L$ (size of the address space).

$$p(n) = (1 \times \left\lceil \frac{1}{L} \right\rceil \times \left\lceil 1 - \frac{2}{L} \right\rceil \times \left\lceil 1 - \frac{3}{L} \right\rceil \times \ldots \times \left\lceil 1 - \frac{L-1}{L} \right\rceil$$

$$\frac{L \times (L-1) \times (L - 2 \times (L - 3 \times \ldots (L - n + 1)}{L^n}$$

$$\frac{L!}{L^n(L-n)!}$$

$$\frac{LP_n}{L^n}$$

The $equation$ expresses the fact that the first node has no duplicate, the second node cannot have the same IP address as the first (L-1/L), the third cannot have the same IP address as either of the first two (L-2/L), and in general the nth node cannot have the same IP as any of the n − 1 preceding nodes. The event of at least two of the n nodes having the same IP address is complementary to all n nodes having different IP addresses. Therefore, its probability p(n) is calculated as follows:

$$1 - p(n)$$

Like similar works such as (Pilar Rios et al 2017), the taylor series can be used for estimation. Using the Taylor series expansion of the exponential function , we can approximate the probability of address conflicts as follows:

$$e^x = 1 + x + \frac{x^2}{2!} + ..$$

provides a first-order approximation for ex for x ≪ 1:

$$e^x \approx 1 + x.$$

To apply this approximation to the first expression derived for $p(n)$, set $x = -{}^{a}/_{L}$. Thus,

$$e^{\frac{-1}{L}} = 1 - \frac{a}{L}$$

Then, replace a with non-negative integers for each term in the formula of $p(n)$ until $a = n - 1$, for example, when $a = 1$,

$$e^{\frac{-1}{L}} = 1 - \frac{1}{L}$$

The first expression derived for p(n) can be approximated as

$$p(n) \approx e^{\frac{-1}{L}} \times e^{\frac{-2}{L}} \times e^{\frac{-1}{L}} \times e^{\frac{-(n-1)}{L}}$$

$$e^{\frac{-1+2+3\ldots(n-1)}{L}}$$

Therefore,

$$e^{\frac{-n(n-1)}{2L}}$$

An even coarser approximation is given by

$$e^{\frac{-n^2}{2L}}$$

which, as the graph illustrates, is still fairly accurate. From the equation above: Assuming a network of 100 nodes in a network capable of supporting 256 valid IP addresses, figure shows the probability of having a duplicate address. From figure 1, the probability reaches at least 50% at less than half the address space.

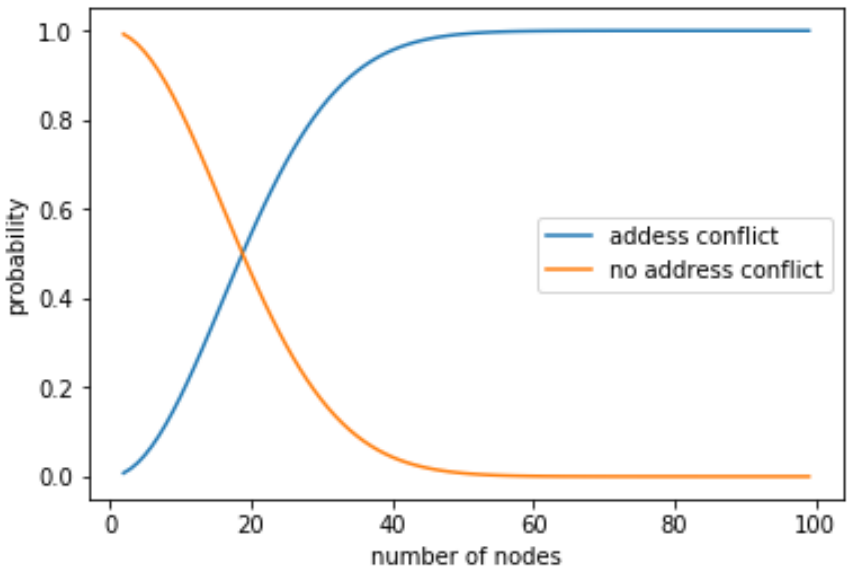

Figure 2.1: Depiction of address conflicts

2.2 IP Address Auto-Configuration Approaches

A number of IP address assignment algorithms have been proposed in the literature. The mechanisms can be classified according to the way they manage the IP address space. There are basically two main categories: stateful and stateless. The stateless approach does not employ a mechanism for managing the IP address space. The number of free IP addresses is not known, hence when new nodes join the network they choose a random IP address and check for availability through a DAD procedure. On the other hand, stateful approaches employ one or more nodes to manage the IP address space. In this section, an analysis of the current address auto-configuration paradigms is given. This lays the foundation for the paradigm shift proposed in the next sub-section.

a) The Stateless paradigm

In stateless protocols, free IP addresses are not known in advance because address allocation tables are not kept. All the network nodes collectively manage the IP address space by participating in duplicate address detection. New nodes generate their own IP addresses from an allowed range and check for possible conflicts. If an address conflict is detected, the new node will repeat the process until a free address is obtained. The process of verifying the uniqueness of the address is called a Duplicate Address Detection (DAD) procedure. Generally, the DAD process is categorised as being either StrongDAD (Perkins et al., 2001) or Weak DAD (Vaidya, 2002). Weak DAD makes use of a key-address combination that must always match if there is no address conflict. Nodes analyse routing protocol packets for signs of address conflicts. StrongDAD is a time-based DAD that

checks if there is an address conflict in a network within a finite-bounded time interval. StrongDAD configures nodes after the DAD procedure has been successfully completed or after a specific time interval called a DAD timeout period.

AIPAC (Fazio et al., 2006), AROD (Kim et al., 2007) and the scheme proposed in Nesargi & Prakash, (2012), are based on StrongDAD (Perkins, et al., 2001). In Nesargi and Prakash (2012), a new node chooses two addresses, a temporary address and the actual address, to use. The temporary address is used only once during the address negotiation phase. The network is then flooded with an address request packet containing the actual address. A node that uses the requested IP address sends an address reply message to defend its address. If no Address Reply (AREPs) are received by the originator after a certain time interval and after multiple tries, the node concludes it can use the chosen address. In AIPAC (Fazio et al., 2006), a new node periodically broadcasts a request message until a reply is received from at least one neighbouring node (initiator). The initiator then performs DAD on behalf of the new node.

AROD (Kim et al., 2007) extends StrongDAD (Perkins, et al., 2001) by including address reservation as a mechanism to reduce the communication overhead. The authors argue that it is difficult to guarantee uniqueness of allocated addresses without performing a DAD. Thus they proposed a distributed auto-configuration scheme that uses address reservation and optimistic DAD. Reserved addresses were introduced to help reduce allocation latency, while the DAD mechanism guarantees the uniqueness of address with much smaller communication overhead than traditional DAD approaches.

Stateless approaches are prone to duplicate IP addresses because of the unpredictable nature of ad hoc networks. When network size increases, the probability of a failed DAD also increases, resulting in delay and communication overhead. Determining the parameters of DAD like the timeout period, is an issue that needs investigation. A static value might not be the best since network conditions are not static. It is therefore imperative to employ adaptive mechanisms that respond to network conditions.

b) Stateful Auto-configuration

There are many variations of stateful auto-configuration but the basic concept is that there is at least one node that is responsible for managing the IP address space. When new nodes join, the node or nodes managing the IP address space can easily issue free IP addresses since they are known in advance. To guard against address leakages, nodes that run stateful auto-configuration synchronise the address allocation tables. IP address auto-configuration approaches following the stateful paradigm can further be classified according to the way they manage the IP address allocation table. The IP address allocation table can either be centralised or distributed. In the case of a distributed allocation table, there are two alternatives: distributing a common table managed by all the nodes, or distributing multiple disjoint allocation tables where each node manages its own pool of IP addresses.

Auto-configuration solutions that use a centralised allocation table must guarantee that the central node is always available and has up-to-date state information to avoid address

leakages and conflicts. Node departures and arrivals should be reflected promptly. In CAC (Güne & Reibel, 2002; Ramakrishnaiah & Reddy, 2016) a central node called the address agent periodically broadcasts verify-packets which contain the address list and a time stamp. Every node checks whether or not it is included in the address list. Traffic to the central node must be well managed so that it does not get overloaded since it is the only one with the responsibility of managing the IP address allocation table. If the address agent (AA) is temporarily unavailable there must be a mechanism of selecting a new AA. In CAC, if a node does not receive any more verify packets from the address agent it assumes that the network is partitioned and elects itself as the new AA.

MANETConf (Nesargi & Prakash, 2012) and the work by Wang et al., (2014) make use of a Distributed Address Allocation Table, i.e. all nodes in the network keep a list of IP addresses that are currently in use. The management of the IP address space is thus distributed to all the nodes. If a node (Requestor) wants to join the network, it has to rely on an already configured node. Then the Initiator selects a free IP address from the address allocation table, and checks for its availability through a DAD procedure.

Prophet (Zhou et al., 2013), uses a novel approach that follows the stateful paradigm but does not make use of an IP allocation table. The basic idea behind Prophet is to predict the allocation table using a function f(n) that is distributed across all the nodes. The first node in the network chooses the function parameters. As other nodes join the network, the function f(n) and a state value to generate IP addresses are passed on to them so that they

can also allocate a new node with IP addresses. In this approach the IP address space is known by all nodes in the network.

In (Bernardos et al., 2005), the authors assessed the PACMAN protocol (Weniger, 2005) in a Wireless Mesh Network scenario. PACMAN is hybrid in nature; it combines PDAD (Weniger & Zitterbart, 2004) with a distributed maintenance of a common allocation table such as the one proposed in MANETConf (Nesargi & Prakash, 2012). PACMAN assumes layer 3 routing and uses cross-layer information from ongoing routing protocol traffic. Based on a pre-defined conflict probability, an estimation of the number of nodes, and an allocation table, the algorithm calculates the size of the virtual address space, randomly selects an address from this space and ensures that the address has not already been assigned according to the local allocation table. The algorithm uses a passively updated state information table that is distributed across all the nodes. The table is passively updated using incoming routing protocol information. The same information is also used to passively detect duplicate IP addresses.

Ancillotti et al., (2009) proposed a DHCP server- dependent auto-configuration scheme for hybrid wireless ad hoc networks. In their proposal, nodes are configured with globally routable addresses using a DHCP-based mechanism in the wired part of the network. New nodes use already configured nodes to act as initiators during the IP address configuration process. The initiator communicates with the DHCP server on the wired part of the network in a multi-hop fashion. Configuration parameters are sent by the server to the new nodes

via the initiator. The AH-DHCP protocol assumes that the gateways are the first to join the network because they can communicate with the DHCP server using their wired interfaces. Without the presence of the DHCP server on the wired part of the network, nodes cannot communicate with the outside world or amongst themselves.

The main challenge of stateful approaches is the design of reliable state synchronisation mechanisms. Frequent state synchronisation messages result in high communication overhead but it helps to reduce duplicate addresses. On the other hand, reducing the frequency of state synchronisation might result in less communication overhead but increase address duplicates. Stable networks might not require frequent state synchronisation.

c) Hybrid Approaches

Hybrid auto-configuration solutions combine characteristics of both stateful and stateless approaches to manage the IP address space. These protocols combine DAD with an address allocation table that is either centrally maintained or distributed. Hybrid Centralised Query-Based Auto-configuration (HCQA) protocol utilises StrongDAD together with a centrally maintained allocation table. PACMAN (Weniger, 2004) combines PDAD proposed in (Weniger, 2003) with a distributed maintenance of a common allocation table.

In Wise-DAD (Mutanga et al., 2008), all the nodes passively collect state information but still perform DAD when a new node wants to join the network. A non-configured node selects one of its neighbouring nodes to act as its negotiating agent (initiator). The Initiator then generates a random IP address from the allowed addresses and checks its allocation table to determine if there is any node in the network that has requested for or is currently using the same IP address. If the address is not known, the initiator then performs a DAD (using an address request message). 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 and the state information (allocation table). 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 synchronised; 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.

2.3 The IP address Allocation Problem

The problem of IP address auto-configuration can be classified as a self-organisation problem. In essence, the idea is to get the nodes to collectively organise themselves without external intervention or central control such as a DHCP server. 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 calls for some very interesting applications in self-orgnisation in computer science.

In schooling fish, while the group as a whole exhibits some coherent global behavior, 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.

This is similar to the configuration of addresses where nodes are expected to self configure while avoiding address conflicts with their neighbours and the network as a whole. The mechanism, by which the global emergent behavior relates to the simple, limited, unplanned and unreliable individual agent activities, is quite compelling to computer scientists and the research society as a whole. For this to work, it is imperative to define rules that govern local interaction of nodes to achieve a global goal. What makes the design of this solution challenging is the fast that there are multiple criteria to adjust to meet multiple objectives. This thesis approaches this problem as a self-organisation problem that has constraints of a multi-criteria decision-making problem. Multicriteria decision making (MCDM) involves deciding the presence of numerous and contradictory criteria.

Traditional solutions took either the stateless or the stateful approach. Stateless approaches are prone to duplicate IP addresses because of the unpredictable nature of ad hoc networks. In essence, they fail to balance the conflicting objectives because when network size increases, the probability of a failed DAD also increases, resulting in delay and communication overhead.

On the other hand, stateful approaches fail to balance the contradictory objectives because of the need to perform reliable state synchronisation mechanisms. Frequent state synchronisation messages result in high communication overhead but it helps to reduce duplicate addresses. On the other hand, reducing the frequency of state synchronisation might result in less communication overhead but increase address duplicates.

Problems for MCDM may range from problems we encounter everyday, such as buying a house or a vehicle , to those affecting entire economy. Nevertheless, even with the diversity, all the MCDM problems share the following standard features (Ahmadvand and Tamalloki, 2017):

i. *Multiple criteria*: each problem has multiple criteria, which can be objectives or attributes.
ii. *Conflicting among criteria:* multiple criteria conflict with each other.
iii. *Incommensurable unit:* criteria may have different units of measurement.
iv. *Design/selection*: solutions to an MCDM problem are either to design the best alternative(s) or to select the best one among previously specified finite alternatives.

There are two types of criteria: objectives and attributes. Therefore, the MCDM problems can be broadly classified into two categories:

(i) Multi objective decision making (MODM).

(ii) (ii) Multi attribute decision making (MADM).

The main difference between MODM and MADM is that the former concentrates on continuous decision spaces, primarily on mathematical programming with several objective functions, and the latter focuses on problems with discrete decision spaces. What makes the problem of address allocation more challenging is the fact the it presents us with multiple objectives and multiple attributes at the same time making it difficult to address it as a simple MCDM problem. Table 2.1 below shows the intertwined relationships between the objectives and the constraints.

	Low	**High**	**Low**	**High**	**not frequent**	**Frequent**
	Low	High	Low	High	High	Low
	High	Low	Low	High	Low	High
			Low	High	Low	High

Table 2.1 : relationship between the objectives and constraints

In general, a multi objective problem can be represented as follows:

$$\begin{cases} \max & f(x) \\ \text{s.t.} & x \in S, \end{cases}$$

The function f(x) refers to n conflicting objectives whilst x is an n-vector of decision variables. In this case the objectives are reduction in latency, reduction in address conflicts and communication overhead. On the other hand our variables are DAD timeout period, number of DAD trials and state synchronisation frequency. All this have to be handled in a dynamic environment which includes among other things, mobility, high network traffic volumes, varying network membership. In optimisation, multiple criteria problems are still an open and challenging area to provide solutions to. Multiple-criteria decision-making (MCDM) evaluates multiple conflicting criteria in decision making conflicting criteria are typical in evaluating options: cost or price is usually one of the main criteria, and some measure of quality is typically another criterion, easily in conflict with the cost. For example, higher value of DAD timeout may lead to high delay, at the same time may lead to low address conflict. On the other hand more DAD trials lead to high communication overhead and high latency but reduces the probability of address conflicts. Frequent state synchronisation leads to high communication overhead, high latency but reduces address conflicts. Such a complex problem should be structured well carefully considering both the multiple criteria and the multiple objectives. In the following subsection, we present a generic solution to the problem. Our solution not only consider the multiple objective and criteria but also the context in which the problem exist. This makes traditional approaches

to multi decision and multi attribute problem inapplicable. The solution proposed below calls for conducting experiments to determine the optimal settings for the multiple criteria identified in the definition of the problem under investigation.

2.4 The Swam Intelligence Inspired IP address Space Management Model

From the analysis of current solutions conducted in the previous sub-section, it is clear that there is no single approach or protocol that is significantly superior to another. This is so because in meeting all the design requirements explained in section 2.1 the following performance metrics goals usually contradict each other: low latency, high probability of address uniqueness, and low communication overhead. For example, the best way of making sure that the allocated IP address is unique is to perform DAD, 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 to get maximum benefits (desirable characteristics) while keeping the costs (undesirable properties) associated with attaining those conditions as low as possible. Therefore, the design must consciously make trade-offs between these contradictory factors. This is easily achieved if the network conditions are predictable. However, the unpredictable nature of wireless ad hoc networks presents challenges to the auto-configuration algorithms.

Contrary to other schools of thought, this thesis argues that for the purposes of IP addressing, a wireless ad-hoc network should be viewed as a flock or a school of fish swimming together toward a certain direction. In schooling fish, interactions among the fish are based on simple behavioural rules that exploit only local information. This information is exchanged directly amongst immediate neighbours or via the environment. In the configuration of IP addresses, nodes must collaboratively perform the duties of DHCP without external input. In addressing this problem, this thesis therefore takes swam intelligence approach.

Swarm intelligence is the discipline that deals with natural and artificial systems composed of many individuals that coordinate their activities using decentralized control and self-organization. It has potential to solve complex problems.

In particular, the discipline focuses on the behavior of social insects such as fish schools and bird flocks and colonies of ants, termites, bees, and wasps. Self-organization, robustness, flexibility and handling unpredicted situations are some of the application areas of such collective and cooperating strategies.

In address auto-configuration, interactions should be limited to localised communication so as to conserve bandwidth but the resultant emergent characteristic should resemble that of a system which is centrally controlled. This is analogous to flocking behaviour in schooling fish or birds. The most well-known examples of systems studied by swarm intelligence are particle swarm optimization (PSO) and ant colony optimization (ACO).

Particle swarm optimization mimics the behavior of fish schooling and bird flocking. PSO is a population-based stochastic optimization strategy with fast convergent speed than general evolutionary algorithms (EAs).

They communicate good positions to each other and adjust their own positions according to their decision. In PSO, a number of simple entities—the particles—are placed in the search space of some problem or function, and each evaluates the objective function at its current location. Each particle then determines its movement through the search space by combining some aspect of the history of its own current and best (best-fitness) locations with those of one or more members

Flocking behaviour in swam intelligent systems defines 3 very crucial characteristics that can be adopted in IP address auto-configuration:

(a) Separation - avoid crowding neighbours : This characteristic talks about the need to separate the responsibility of steering the flock to all the members of the flock. The neighbours have to operate independently from yet achieve the global desire goal exhibiting high levels of cohesion as though there was central control. This property can be achieved by distributing the address allocation protocol and delegating the responsibility of address configuration to the whole network or a selected set.

(b) Alignment – This property calls for flock members to steer towards average heading of neighbours. Although, separated, there is need for individual members of the flock to align themselves with others so as to achieve the desired goal of moving towards a certain direction at the same speed without a central leader yet not bumping into each other. To achieve this, an address autoconfiguration protocol should implement rules that adapt to changes within the network.

(c) Cohesion - steer towards average position of neighbours : This rule tries to make the members of the flock mimic each other's course and speed. If this rule is not used, the members would bounce around a lot and not form the beautiful flocking patterns that can be seen in real flocks.

The proposed model is given in Figure 2.1. A full description of the components of the proposed model are given in the following sub-sections.

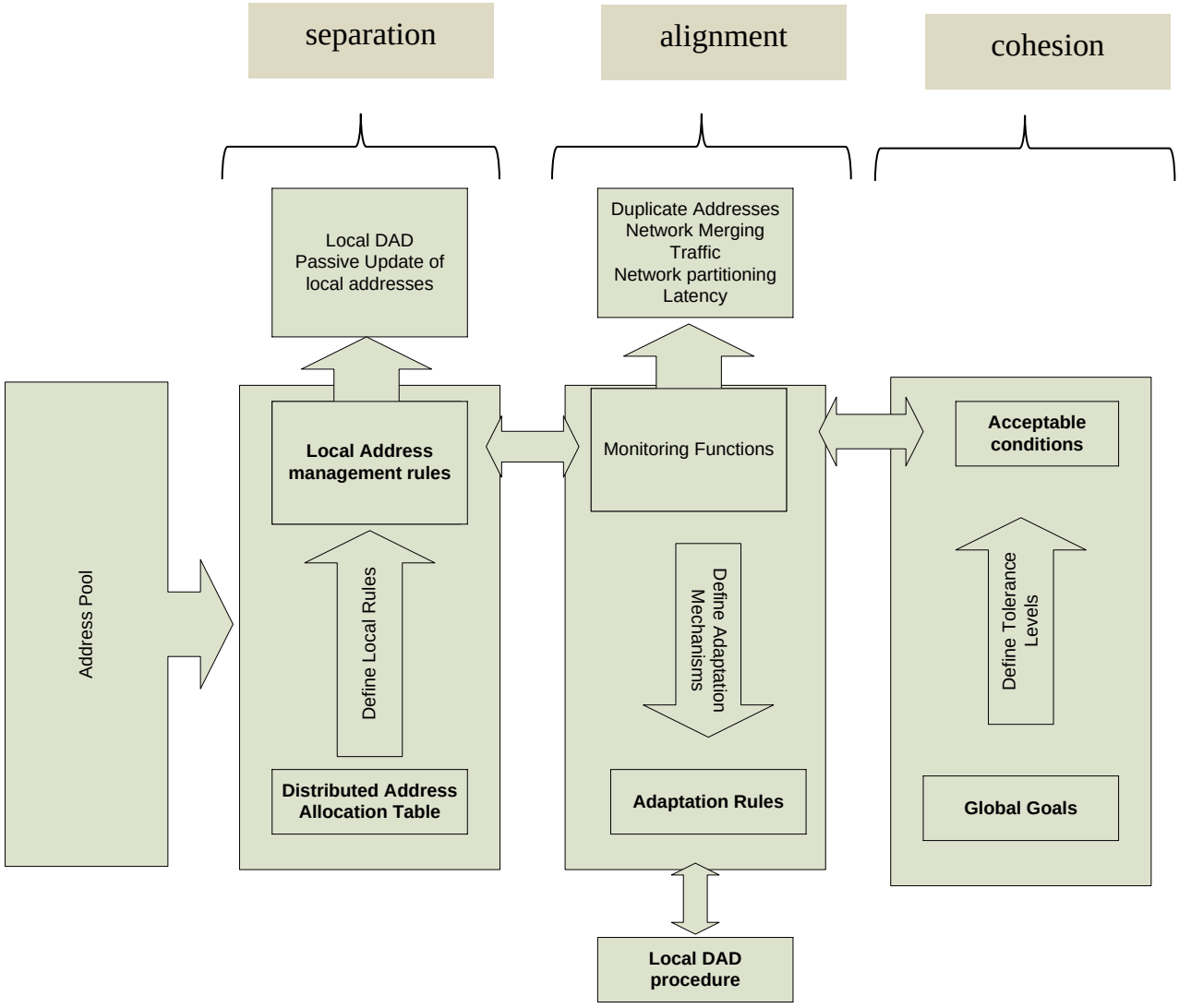

Figure 2. 1: The adaptive Model

a) Separation: IP addresses should be distributed to all nodes

To achieve the property of separation, the address allocation processes should be distributed rather than centralized. The distributed nature of swarm intelligent systems maximizes the overall system dependability by removing critical challenges such as single point of failure, bottlenecks and unbalanced traffics. To achieve separation, the first task therefore is to define mechanisms of managing the IP address space in this fashion. To achieve this, we need to explicitly define a mechanism that delegates the responsibilities of assigning IP addresses to all the nodes in the network. This scenario presents us with multiple objectives and multiple attributes at the same time making it difficult to address it as a simple MCDM problem. There is, therefore, a need to explicitly define how this decentralised system will be managed, that is, localised behaviour rules or functions that, if applied on all nodes at a microscopic level (within their local neighborhoods), automatically lead to the desired network behaviour at a macroscopic level. This mechanism should be capable of striking a balance between the multiple objectives and multiple attributes. Network nodes must have only a local view of the network and interact with their neighbours as much as possible whilst the whole network follows the desired global property synonymous to schooling fish or flocking birds. This reduces both allocation time and minimises communication overhead since communication will be done locally. To achieve this, there is a need to define the following building blocks of a framework that follows the adaptive paradigm (Figure 2.1).

i. Mechanisms, rules or functions for managing the local IP addresses: These rules can be implemented as functions that govern the behaviour of individual nodes in their own neighbourhoods. The behaviour of nodes should yield the desired global properties if applied consistently. It must be clear how the IP addresses will be managed locally without adversely affecting the realization of the desired global goal of ensuring that there is cohesion. Locally managing IP addresses, among other things, reduces communication overhead generated by the protocol. The local management rules should also take in account the fact that there are multiple conflicting objectives to be achieved under very tight constraints. Managing this constraint improves the scalability of the auto-configuration protocol. Reduction of communication overhead and improving scalability are some of the key global properties of address auto-configuration requirements outlined earlier in section 2.2. To realise this function, there is a need to investigate the best DAD parameter configurations such at DAD timeout.

ii. Functions or rules governing the delegation of the responsibilities of assigning IP addresses to all the network nodes in such a way that the rules defined in (i) above can be applied with ease: The functions should delegate the responsibilities in a manner that achieves the desired global goals. The responsibility of address allocation can be assumed by either all or a set of selected nodes. Initial investigation identified the importance of DAD to guard against address duplicates. However, DAD is a network-wide broadcast that can

result in high communication overhead. Furthermore, it is against one of the key principles of swam intelligence. It is thus important to establish the best configuration parameters for DAD. The investigative experiments carried out in chapter 3 established the best configuration parameters for DAD .

b) Cohesion : State information should not be explicitly coordinated or synchronised

Although the first property advocates for separation, the resulting emergent behavior should exhibit high levels of cohesion amongst the entities of the network. In address auto-configuration, to achieve cohesion, there is need to synchronize state information. However, this contradicts objectives of having low communication overhead and delay. Therefore, this thesis is of the opinion that, state information should not be explicitly coordinated because of the high communication overhead that may occur. To achieve this, three rules for managing state information synchronization are proposed:

i. All possible states and how they affect network behaviour must be defined. Mechanisms for how to respond to each state can then be defined to react to the identified state changes. Although a lot of work has been done in developing new auto-configuration protocols it is still not clear how network conditions affect the auto-configuration process. An investigation of how network conditions affect the auto-configuration process is presented in chapter 4.

ii. The level of state information inconsistencies that can be tolerated must be defined: When the network increases in size, coordination can be very difficult. Also, due to the unpredictable nature of ad hoc networks, explicit coordination can be bandwidth- consuming if network conditions change frequently. The network partitioning and merging mechanisms proposed in this work can tolerate temporary occurrence of network merging and partitioning, unlike other proposals in the literature. Details of these mechanisms are given in chapter 5.

iii. State information must be passively synchronised: Mechanisms of passively obtaining state information must be defined. This can be achieved by using routing protocol control packets such as hello messages.

c) Alignment: Address auto-configuration protocols should adapt to changes

Due to the unstable nature of wireless ad hoc networks the environment in which the nodes operate may change unexpectedly. There is need to align what is happening at a microscopic level to the overall goal of the protocol. To achieve this, this thesis proposes that an IP address auto-configuration protocol should adapt to different triggers to change. This increases robustness of IP address allocation schemes. The desired performance of an IP address allocation scheme is measured at a macroscopic level, hence there is a need to monitor if the distributed address management is achieving the desired global goal. To

avoid address leakages, acceptable levels of non-coordination defined in (b) above should be monitored and corrective action taken if the need arises.

Other scenarios that require protocols to employ a monitoring mechanism include the merging of two or more independently configured networks, network partition, and the exhaustion of local IP addresses. All unforeseen occurrences need to be monitored and corrective action taken. This thesis therefore argues that an IP address protocol for wireless ad hoc networks should have the following functions for the purpose of adapting to different triggers for change.

i. A performance monitoring mechanism that monitors the changes that might require corrective action: This can be implemented using monitoring algorithms or functions that can either be proactive or reactive in nature. Increase in address allocation latency, communication overhead, network merging, security threats, and address duplicates are some of the performance metrics that should be monitored. Due to bandwidth limitations within the wireless ad hoc environment, this thesis proposes that the design of the monitoring mechanisms must be bandwidth conscious so as to minimise communication overhead. For the network to be able to respond to changes, the protocol must first detect the changes. In chapter 5, the algorithms to monitor network conditions are presented and analysed in chapter 6. The algorithms proposed in chapter 5 passively monitor the network environment without burdening the network with additional

data packets. This was done to enhance the scalability of the auto-configuration mechanism.

ii. A response management strategy that takes action should a change that requires nodes to behave differently be detected by mechanisms employed in (i) above: The nodes should take action that is relevant to the changes observed. A DAD procedure should also be part of an IP address auto-configuration scheme. It can be defined under the adaptation rules to guard against erroneous address allocation. If the current settings or configurations of the protocols are no longer yielding the desired results, it is important to determine the optimal values that will give the desired results. For example, it was established that the optimal value of the DAD timeout period and the number of DAD trials depend on network conditions (see chapter 3). Network conditions might change any time, hence these values must be calculated based on the network conditions. Setting static values will not result in an optimal protocol since network conditions can change at any time. The IP address auto-configuration protocol presented in this thesis responds to changes in node mobility, high traffic volume and network size. In addition, in chapter 5, this thesis outlines mechanisms of responding to both network merging and partitioning. Gradual merging was proposed as opposed to sudden merging which consumes a lot of bandwidth.

2.5 Chapter Summary

Given the constraints in the wireless ad hoc networking environment, managing the IP address space is not an easy task. Solutions proposed in the literature exhibit a number of limitations in the face of changes in network conditions. This thesis argues that the current paradigms are not suited for wireless ad hoc network. We therefore advocate for a paradigm shift. This chapter presented an approach to managing the IP address space in Wireless Ad hoc networks. This approach proposes the distribution of IP addresses and defines local management rules for the distributed addresses. The proposed model monitors the environment for conditions that are known to affect address configuration and to adapt where possible. Studies on how different networks affect auto-configuration are given in chapters 3 and 4. Protocols following the paradigm proposed in this chapter are likely to be more robust than both stateless and stateful approaches. The biggest research challenge is to come up with the best building components for constructing the monitoring and adaptation modules. Chapters 3, 4 and 5 investigate further and propose an address auto-configuration protocol based on the findings.

Chapter 3

Determining the optimal DAD configuration parameters

3.0 Introduction

Guaranteeing address uniqueness is one of the most important requirements that an address auto-configuration protocol should achieve. The previous chapter concluded that it is difficult to guarantee address uniqueness without performing DAD. DAD is therefore an important building block of address auto-configuration (Dart et al., 2015).

Despite DAD being an important part of address auto-configuration no work has been done to establish the configuration parameters such as DAD timeout and optimal number of DAD trials required during the auto-configuration process. To get optimal performance, the DAD parameters should be investigated (Rana et. Al, 2017). As part of implementing the separation property of the system components such an investigation is important to achieve the cohesion property.

When nodes perform DAD they generate their own IP address and broadcast a request packet and set a timer (DAD timeout). When the DAD timeout expires, before any node using the requested IP address responds the new node configures itself. If the DAD timeout period is not long enough, the new node configures itself before the node using the requested address can defend its IP address. On the other hand, if the DAD timeout period

is unnecessarily long, a long delay might be experienced because the new node will only be able to configure itself after the timeout has expired. In the literature, some solutions resort to repeating DAD two or three times to guard against message losses that might result in assigning duplicate addresses.

This chapter describes an investigation into the optimal DAD timeout period for the address auto-configuration model presented in the previous chapter. DAD is an important building component of the model proposed in the previous chapter. Before DAD can be used as a building component of the model, it is thus imperative to carry out this investigation.

The rest of the chapter is presented as follows: Section 3.1 gives a brief literature review on DAD based address configuration whilst section 3.2 outlines the setup of the investigative experiments. Section 3.3 presents the simulation results and section 3.4 concludes the chapter.

3.1 DAD-Based Auto-configuration

This section starts by giving a review of DAD-based auto-configuration protocols to lay the foundation of the investigation. The review given in this section was done to establish the current knowledge on DAD configuration parameters.

Generally, DAD-based auto-configuration protocols do not maintain any allocation table. The nodes generate their own IP addresses and check for possible conflicts through a Duplicate Address Detection procedure. Hence most of the research in this paradigm is aimed at optimising the DAD mechanism. If a conflict was detected, the new node would repeat the process. Because of this, the DAD procedure is indisputably the cornerstone of the stateless paradigm. Generally, the DAD process is categorised as being either StrongDAD or Weak DAD. StrongDAD is a time-based DAD that checks if there is an address conflict in a network within a finite bounded time interval. StrongDAD configures nodes after the DAD procedure has been successfully completed or after a specific time interval (DAD timeout period). Weak DAD is used for the purpose of detecting IP address conflicts by making use of a key-address combination that must always match if there is no conflict in the network. When a node receives a routing control packet it compares the address and key contained in the packet with those that appear in its routing table.

A weak DAD is usually termed optimistic DAD since it configures the new node before the DAD procedure is complete. It assumes that the DAD procedure will be successful, hence the name optimistic DAD. Even if the DAD is not successful, unicast communication can still take place without any problems since the nodes use the key-address combination

to identify the origins or destination of a packet. However, Weak DAD does consume a lot of bandwidth and requires modifications to the routing protocol. These limitations make it difficult to for it to meet the requirements of auto-configuration protocols outlined in chapter 2.

In StrongDAD auto-configuration, a node randomly selects an IP address and checks whether or not it is utilised in the network using a DAD procedure. In fact a new node chooses two addresses: a temporary address and the actual address to use. During the IP address negotiation process, new nodes use temporary IP addresses. The temporary address is not verified for uniqueness. The network is flooded with an address request (AREQ) message containing the selected address. A node using the requested address defends it by replying with an address reply (AREP) message. 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. StrongDAD (Perkins, 2001) was tested using a DAD timeout period of 1.8 seconds and was seen to result in latency of more than 5 seconds. A total of 3 DAD trials were also used to guard against message losses.

Other protocols that used StrongDAD include Fernandes & Moreira, (2013), Wang et al., (2014), AIPAC (Fazio et al., 2016), and AROD (Kim et al., 2007). In AIPAC a new node periodically broadcasts a Send Request message until a reply is received from at least one neighboring node (initiator). The initiator selects an address at random among the allowed

addresses and sends, in broadcast, a Search_IP packet. The address selected is specified in the packet. Any node receiving this packet checks 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 procedure is restarted, and a new address is selected. Conversely, if no reply is received for a given time interval (DAD timeout of 1.8 seconds), the Initiator sends the Search_IP packet again (2 DAD trials), in order to guard against possible errors in wireless channels. If neither reply arrives, it means that the address is not used yet. Then the Initiator notifies the Requestor with the NetID of the network and the IP address that it has to use.

In Wise-DAD (Mutanga et al., 2008) nodes maintain state information but still performs DAD before a new node is admitted. The new node selects only one of its neighbour's node to act as its negotiating agent (initiator). The initiator then generates a random IP address from the allowed addresses and checks in its allocation table if there is no node in the network that has requested for or used the same IP. If the address is not known, the initiator then performs a DAD (using an address request message). All nodes receiving an address request packet update their tables and add their IP addresses to the packet before broadcasting it. Allocation tables are not actively synchronised; they are used only as an estimate of the state information. The DAD timeout used in Wise-DAD is 1.8 seconds and only one DAD trial is utilised, since there is an estimate of the state information to check for address duplicates before DAD is performed.

3.2 Experimental Setup for the Investigation of the Optimal DAD timeout configuration

This section presents experiments conducted to determine the optimal DAD configurations to be used on the model presented in the previous chapter. The optimal DAD timeout period and the number of DAD trials have not, to the best of our knowledge, been investigated before now. The two parameters are of paramount importance in the model proposed in this thesis because they directly affect the address uniqueness and scalability of the protocol. In the this subsection, we present the description of the experimental Setup for the Investigation of the Optimal DAD timeout configuration.

i. Routing Protocol

Although no routing protocol traffic was exchanged amongst the nodes, all nodes were configured to use the Dynamic Source Routing (DSR) protocol. The simulated DAD protocol was implemented a network layer protocol. We first had to verify the correctness of broadcast (both multi-hop and one-hop) implementation by first running the simulation for 10, 15 and 30 nodes separately. The results show that both multi-hop and one-hop broadcast were correctly implemented.

ii. Physical Data Link Layer Model

To allow for symmetric communication, nodes were configured to use omni-directional antennas. This is important for broadcasting a signal to all directions or when listening for signals from all directions.

iii. 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.

iv. 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.

v. Address configuration

In the experimental setup, a DAD procedure similar to the one proposed in Perkins et al., (2001) was used. Up till now, the traditional DAD protocols have not used the concept of initiator and requestor. In this new approach, a new node relies on a configured node to solicit for an address on its behalf. The two new concepts have been introduced to guard against two nodes using the same temporary IP address. Using this method, when a new node sends a ’request to join’ message to its immediate neighbours; the first neighbour to

respond becomes the new node's initiator. The initiator replies with 'initiator_available' message and the new node will send an acknowledgement message.

The initiator then chooses a random IP address from a predetermined range and broadcasts an Address Request message. Any node using the requested address will defend its address by an Address Reply message to the initiator; otherwise it will just forward the message.

If no response is received after the set DAD timeout period, the initiator broadcasts the Address Request message again for a predetermined number of DAD trials to guard against time delays and message losses. If, after the set DAD trials, no response is received, the initiator will send an address_packet to the new node. In a bid to establish the optimal DAD trials, the number of the DAD trials were varied in the experiments.

A. Performance Metrics

Handling of network merging and partitioning is not within the scope of this investigation since it only seeks to assess the effects of DAD timeout period on the performance metrics listed below:

(a) Address Allocation Latency

This refers to the average time taken for a node to be assigned an IP address. The address assignment process must be done in the minimum time possible. DAD- based address auto-configuration protocols only configure IP addresses after the expiry of the DAD timeout period. A shorter DAD timeout period will always result in low latency. However, a shorter DAD may also result in address conflicts. A high value of DAD timeout period may result

in unnecessary latency. It is therefore imperative to establish the optimal value for DAD timeout.

(b) Communication Overhead

This refers to the average number of address assignment packets generated and forwarded by each node during the address assignment procedure. A good IP address auto-configuration protocol should use as few messages as possible and the communication should preferably be local. Network-wide and periodic flooding should always be avoided. If a protocol uses a lot of DAD trials, communication overhead increases, hence one of the goals of this investigation was to determine the optimum number of DAD trials.

(c) Address duplicates

This refers to the average number of address conflicts in the network. A good scheme should minimise the probability of having more than one node using the same IP address. A short DAD timeout period may affect the delivery of vital address auto-configuration packets, hence increasing the likelihood of configuring duplicate addresses.

(d) Latency for IP Conflict Message

This refers to the time required for a node to receive a conflict notification message if an address duplicate is detected. A new node configures itself with the chosen IP address when the DAD timeout period expires. If the DAD timeout period is very short a node may configure itself with a duplicate address before the other nodes can defend their IP addresses. The DAD timeout period should therefore be long enough to allow network

nodes to defend their IP addresses. It is therefore imperative to investigate the average latency for the IP conflict message.

3.3 Simulation Experiments on the Investigation of optimal DAD Configurations

i. Experiment 1: Determining the optimal DAD timeout period

The purpose of this experiment was to determine the optimal DAD timeout period by investigating the effect of different values of DAD timeout on latency, address uniqueness, and communication overhead. The nodes were spread over a rectangular 2000m x 2000m flat area for 6000 seconds of simulation time. The simulation parameters for this experiment are shown in Table 3.1.

Table 3. 1 Simulation parameters for experiment I

Parameter	Environment
Number of nodes	30, 60, 90, 120
DAD timeout (seconds)	0.1,0.2, 0.4 , 0.6 … 2
Node arrival rate	1 node / 30 seconds
Address Range	8-bit (256)
DAD trials	1
Simulation time	6000 seconds

a. Effect of DAD timeout period on latency

Figure 3.1 shows that the DAD timeout period is directly proportional to the length of the configuration process. This is due to the fact that configuration only takes place after the DAD timeout period has expired.

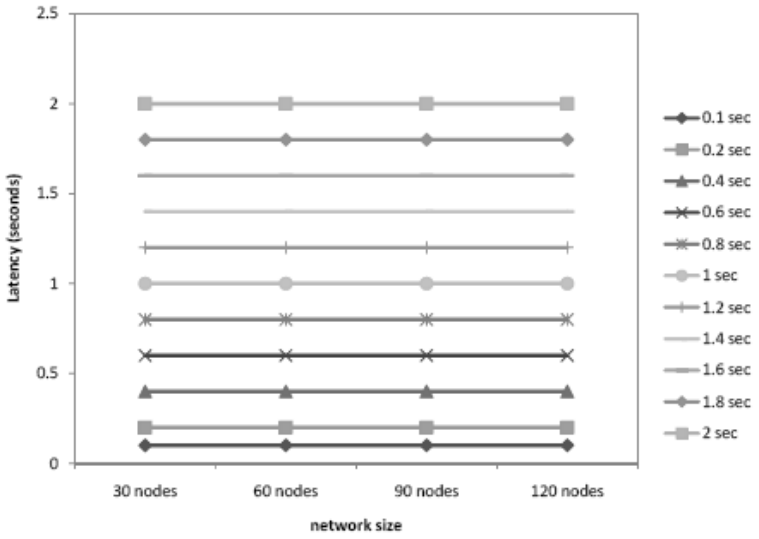

Figure 3. 1: The effect of DAD timeout on latency

b. Effect of DAD timeout on address uniqueness

The results obtained (Figure 3.2) show that the numbers of address duplicates are affected by DAD timeout period. Low values of DAD timeout period result in more address duplicates than larger values of DAD timeout. This can be attributed to the fact that some nodes were not able to defend their IP addresses before the DAD timeout expired, leading to address duplicates. However, as the DAD timeout period was increased, nodes were able to defend their IP addresses; hence, address duplicates decreased. At DAD timeout of 1

second and above, the number of address duplicates did not change significantly except on the 120 node network. It can be concluded that at this value all nodes were able to defend their IP addresses although the same cannot be said for a 120 node network. Any value more than one second was therefore more than the required time for a node to defend its IP address.

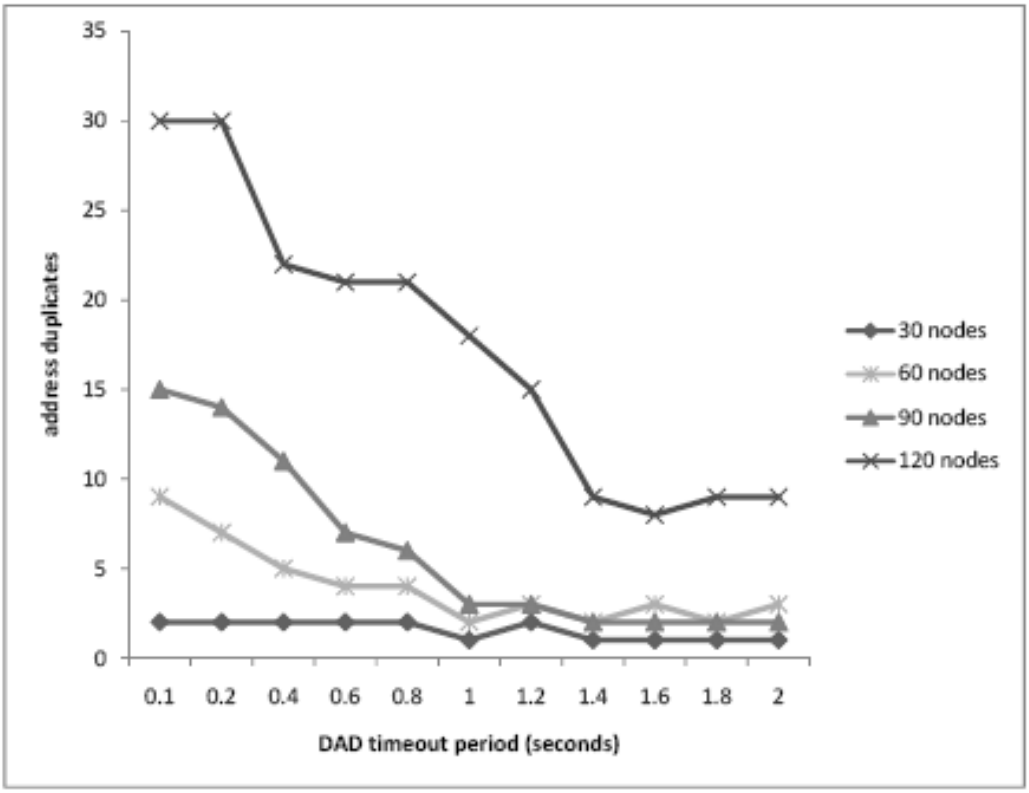

Figure 3. 2 The effect of DAD timeout on address duplicates

c. Effect of DAD timeout on communication overhead

Figure 3.3 shows that communication overhead did not change significantly as the DAD timeout was varied. Interesting to note is the fact that at DAD timeout period of 1 second and above, communication overhead slightly increased. At the same value, Figure 3.2 also shows that address duplicates decreased. It can be concluded that the increase in communication overhead was due to the fact that nodes were able to defend their IP addresses, hence Figure 3.2 showed a decrease in address conflicts. From Figure 3.3, we can conclude that DAD timeout period does not have an effect on communication overhead.

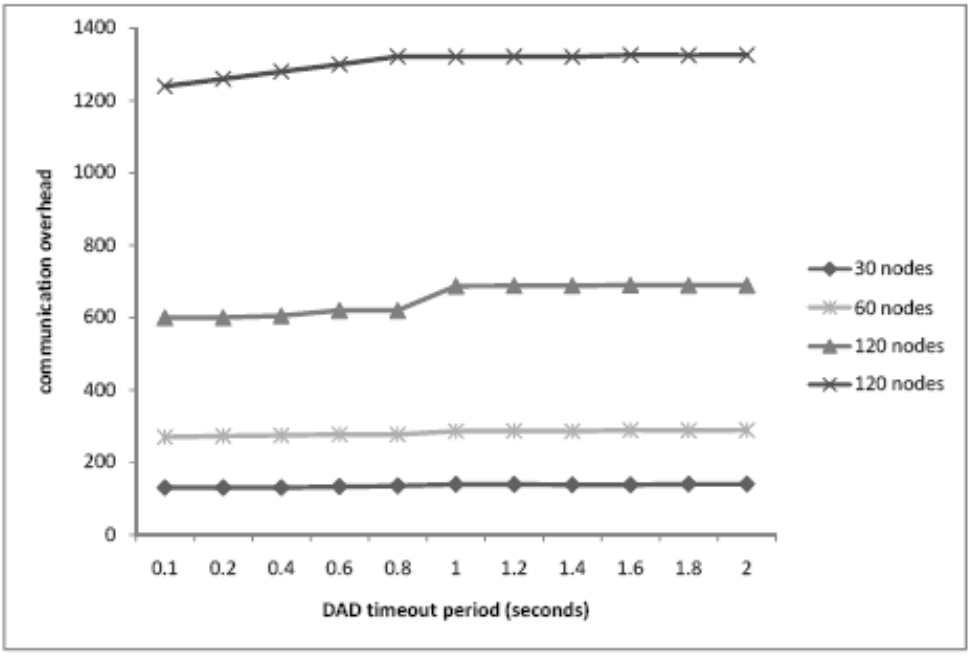

Figure 3. 3: The effect of DAD timeout on communication overhead

ii. Experiment 2: Determining time required for conflict message delivery

The purpose of this experiment was to investigate the time that is required for an address conflict to reach the new node. This thesis argues that the time required for an IP address conflict message to be delivered should be the minimum value that a DAD timeout should use. A node with a duplicate address was created and the time required to detect the duplicated address was measured. Network size was varied because different network sizes might result in different delivery times due to scalability issues. DAD timeout was set to a very high value to give enough time for the conflict message to reach the new node (5 seconds).

Table 3. 2 : Simulation parameters for experiment 2

Parameter	**Environment**
Number of nodes	30, 90, 120
DAD timeout	5 seconds
Address Range	8-bit (256)

The results presented in Figure 3.4 show that the time taken for a conflict message to be delivered is at least 1 second. 120 nodes recorded slightly below 1.2 seconds in latency. These values help in determining the best value for DAD timeout period when designing

an address auto-configuration protocol. From the results we can conclude that using a value which is less than 1 second will result in some nodes not being able to defend their IP address. On the other hand, using a DAD timeout value that is more than 2 seconds will result in unnecessarily high latency.

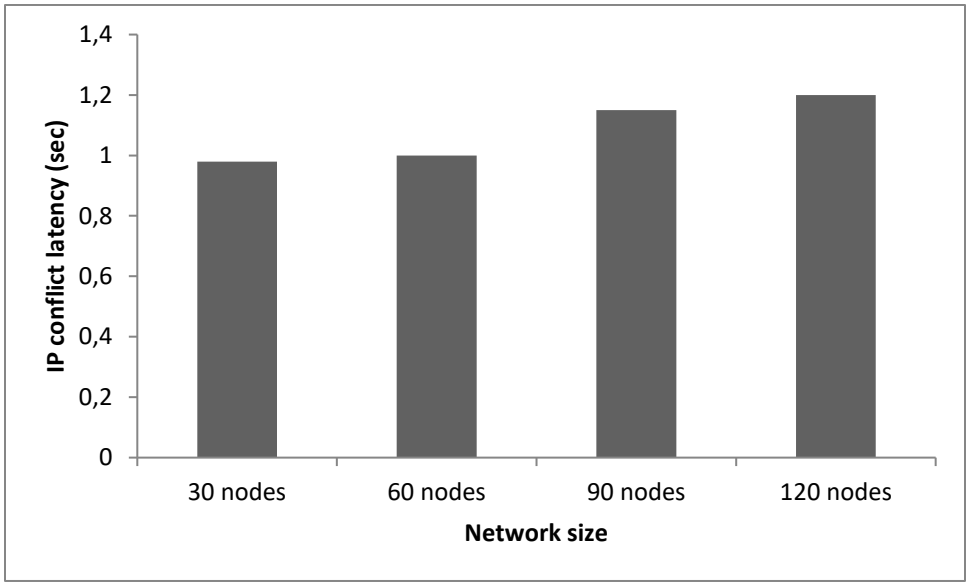

Figure 3. 4: Determining time required for conflict message delivery

iii. Experiment 3: Determining the optimal number of DAD trials

The purpose of this experiment was to determine the optimal number of DAD trails by investigating the effect of the number of DAD trials on the performance of DAD. Some DAD-based protocols use varying DAD trials per requested address before a new node can configure itself. After a new node generates an IP address it sends a DAD message with

the requested address and waits until the timeout period has expired. To guard against message losses the new node will send the DAD message again even if it did not receive a conflict message. The number of times that the message is broadcast varies with protocols. For example, in StrongDAD, three trials are used whilst in Wise-DAD only one trial is used.

Table 3. 3 : Simulation parameters for experiment 3

Parameter	**Environment**
Number of nodes	30, 90, 120
DAD timeout period	1 second
DAD trials	1,2,3
Address Range	8-bit (256)
Simulation time	6000 seconds

a. Effect of DAD trials on latency

Figure 3.5 shows that the number of DAD trials and latency were seen to be proportional to each other. This is due to the fact that each additional trial brings more delay, hence the more the trials the more the latency.

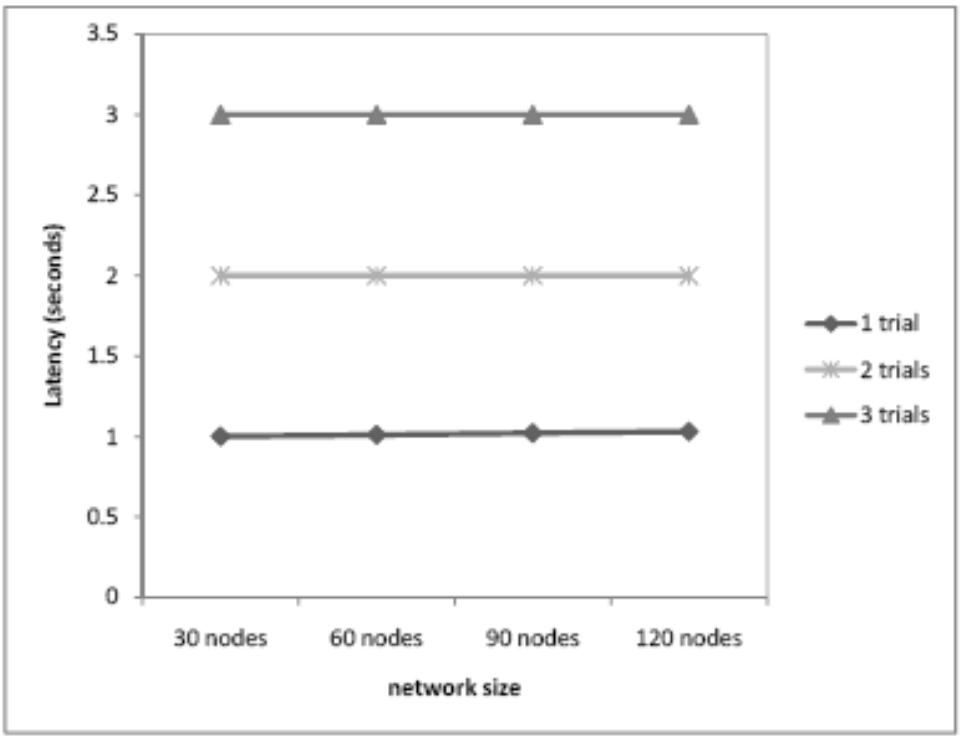

Figure 3. 5: The effect of DAD trials on latency

b. Effect of DAD trials on communication overhead

The communication overhead generated by the configuration process increased with DAD trials. This is due to the fact that each DAD trial generates its own overhead. However, the rate of increase of communication overhead is proportional to the number of nodes.

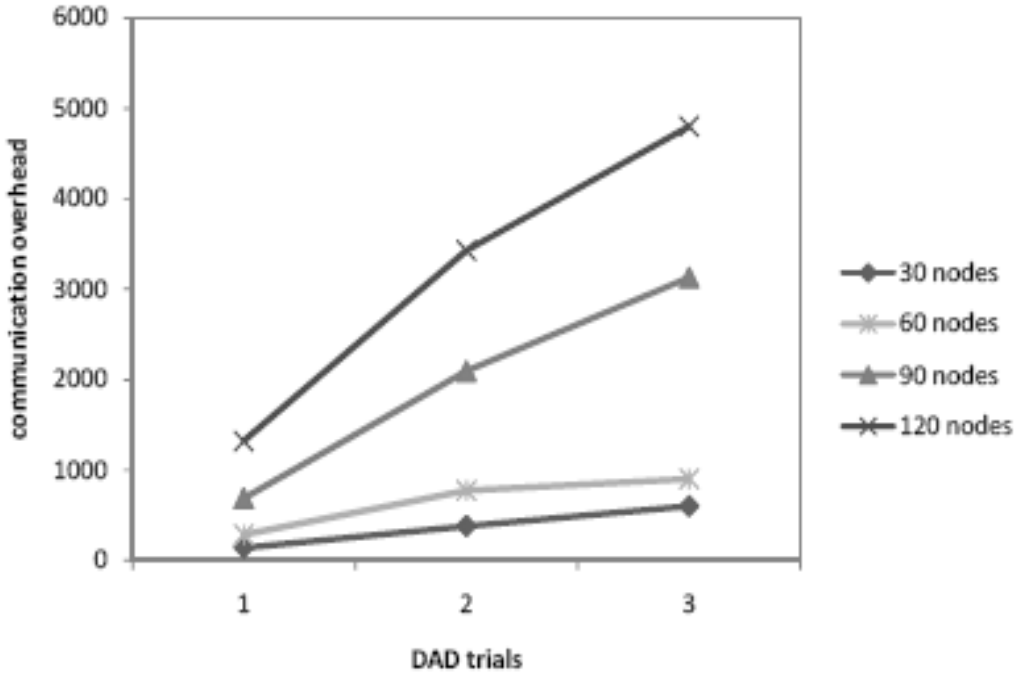

Figure 3. 6: The effect of DAD trials on communication overhead

c. Effect of DAD trials on address uniqueness

The results shown in Figure 3.7 show that address duplicates were not significantly affected by the number of DAD trials. This can be attributed to the fact that the DAD timeout period of one second that was used was long enough for address conflicts to be reported as shown in Figure 3.2 ,hence the conclusion that this is the optimal DAD timeout period can be inferred. The increase in communication overhead at a timeout of 1 sec that is shown in Figure 3.6 also suggests that more nodes were able to defend their IP addresses, hence generating more packets.

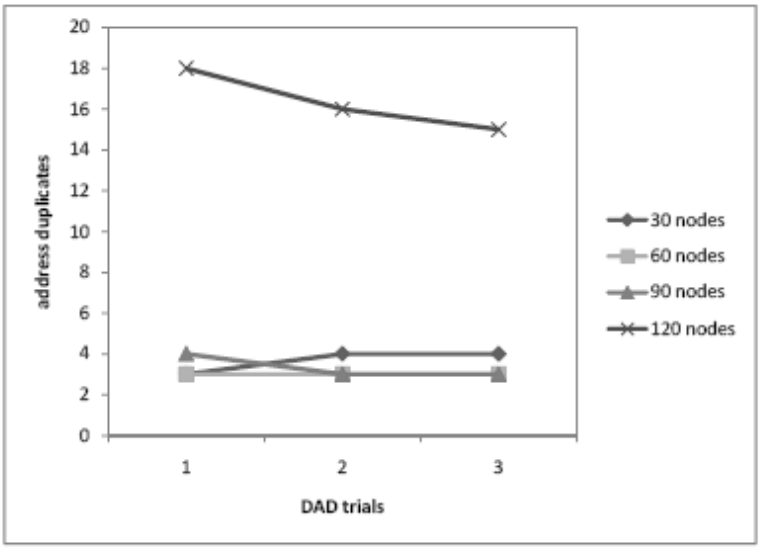

Figure 3. 7: The effect of DAD trials on address duplicates

3.4 Chapter Summary

Many address auto-configuration protocols utilizing DAD have been proposed in the literature. In fact, DAD has become a de facto building block for address auto-configuration, but very little has been done to investigate the best way of setting DAD parameters. The experiments described in this chapter investigated the optimal configurations surrounding DAD. The investigation was an attempt to get the optimum DAD timeout period. The chapter also investigated the relationship between DAD timeout period and network size. The results obtained show that a DAD timeout period of 1 second is the optimal one. It is however imperative to test this timeout period on larger networks. DAD timeout period was found to affect both communication overhead and latency. The results described in this chapter were used to design the solution presented and evaluated in chapters 5 and 6 respectively.

Chapter 4

Effect of Network Traffic and Mobility on Address Auto-configuration

4.0 Introduction

The unpredictable nature of wireless ad hoc networks makes it difficult to have protocols that work effectively all the time. The model proposed earlier in chapter 2 identified the need for the auto-configuration protocol to monitor network conditions. It is known that conditions such as node position, network topology, and mobility have been found to have an effect on routing protocol performance (Kumar et al., 2015; Sibeko et al., 2015; Varshney et al., 2016). It is, however, not clear how the same conditions affect the address auto-configuration process. In order to build mechanisms that take into account network conditions, it is imperative to investigate how different network conditions affect the address auto-configuration process.

Because one cannot predict network conditions, it is challenging to tailor-make protocols for them. However, there are basic conditions that can be assumed to exist in a network most of the time. It is imperative that address auto-configuration protocols are able to detect deviations from the normal and adapt accordingly. While much effort has been put into the development of new IP address auto-configuration protocols for wireless ad hoc networks, very little has been done in testing how different network conditions affect the performance of these protocols. The dynamic change in node membership of ad hoc

networks means that more sophisticated protocols are desirable. The model proposed in chapter 2 requires that nodes monitor their environment for any conditions that may affect the functioning of the IP address auto-configuration protocol. Of interest is the effect of network traffic and node mobility on address auto-configuration. Network traffic will always be present in any network, hence it is important to test how protocols perform under different network traffic conditions. There have been studies on the effect of different types of network traffic and node mobility on routing protocol performance (Al-Maashri & Ould-Khaoua, 2006; Kumar et al., 2015; Rao & Singh, 2015; Tan & Kim, 2014; Thriveni et al., 2013) but it seems no such work exists for address auto-configuration. The performance of an auto-configuration mechanism can be evaluated using a number of characteristics (Schoeneich & Sutkowski, 2016).

This chapter, therefore, establishes the effects of network conditions on the address auto-configuration protocol. The results of this investigation were used as a building block of the address auto-configuration solution based on the model proposed in chapter 2.

The remainder of this chapter is organised as follows. In section 4.1, the design of DAD and evaluation criteria are given. A description of the setup of the experiments conducted to test the effect of network traffic on DAD is presented in section 4.2. Section 4.3 presents the results of simulation experiments on the effect of mobility of DAD. Section 4.5 concludes this chapter.

4.1 Experimental Setup: Effect of network traffic on Address Auto-configuration

To test the effect of network traffic on address auto-configuration, an address auto-configuration mechanism similar to the one used in chapter 3 was simulated. For the purposes of the experiments, a DAD procedure similar to the one proposed in Perkins, (2001) was used. Slight modifications were made. To guard against two nodes joining at the same time and requesting for the same IP address, the concept of initiator and requestor was used. Details of the procedure are explained in section 3.2. In the simulation experiments conducted, the following performance metrics were utilised for the evaluation of DAD.

a) Latency

This refers to the average time taken for a node to be assigned an IP address. The address assignment process must be done in as short a time as possible. The literature has shown that mobility and different types of traffic affect routing (Nitnaware, 2016). Latency is one of the metrics that are negatively affected by different network conditions (Wang et al., 2015). It is therefore imperative to determine the effects of network traffic and mobility on latency during address auto-configuration.

b) Communication Overhead

In a wireless ad hoc network, the number of control packets must be limited. This is mainly due to the bandwidth limitations that characterise the wireless ad hoc environment. In the

experiments, network traffic generated by the nodes was not classified as communication overhead.

c) Address duplicates

The literature has shown that node mobility in wireless ad hoc networks affects packet delivery ratio (Pondwal & Saini, 2016). Poor delivery of address auto-configuration packets has also been found to increase the number of duplicate addresses. It is expected of an address auto-configuration protocol to minimise the probability of having more than one node using the same IP address. In this chapter, the idea was to determine if the network traffic and node mobility have any effect on the number of duplicate addresses recorded in the network.

4.2 Simulation Results: Effect of network traffic on Address Auto-configuration

This sub-section presents the results of the investigation of the effects of network traffic on Duplicate Address Detection. Network size and the type of network traffic were varied to gain a comprehensive insight into the effect of network traffic on address auto-configuration. Address uniqueness, communication overhead, and latency were used for analysis. In the simulation, an evaluation of the performance of the DAD procedure under three different types of network traffic, namely, Pareto, Exponential, and Constant Bit Rate (CBR), was conducted.

a) **The effect of network traffic on DAD**

The purpose of this experiment was to investigate the effect of network traffic on duplicate address detection on different network sizes. In each case, 20 nodes were introduced into an already existing network that had nodes already communicating. No address duplicates existed before the new nodes were introduced. Network traffic following the CBR Model was introduced prior to any new nodes joining the network. Packets size was set at 64 bytes generated at a constant rate of 2 kb/s. The packet inter-arrival time was set at 600ms. The holding time of the model follows a Pareto distribution with a mean of 300s and a shape parameter of 2.5. Configuration delay, communication overhead, and the number of address conflicts were recorded for analysis.

i. Effect of Network size on address uniqueness

Figure 4.1 shows the number of address duplicates against the number of nodes. When the network had no traffic the number of address duplicates was lower than in the presence of network traffic. As the number of nodes increases, the difference between the duplicates recorded in the two experiments increases. The number of duplicates recorded in the presence of network traffic increases at a faster rate than in the absence of network traffic. This can be attributed to the fact that network traffic may have negatively affected the delivery of address allocation packets before the expiry of DAD. If address allocation packets are not delivered address duplicates are bound to occur since nodes will not be able to successfully defend their IP addresses. In StrongDAD, once the DAD timeout period expires, the requested IP address is configured. When there is network traffic the DAD

timeout might expire before an address allocation packet reporting an address conflict is received.

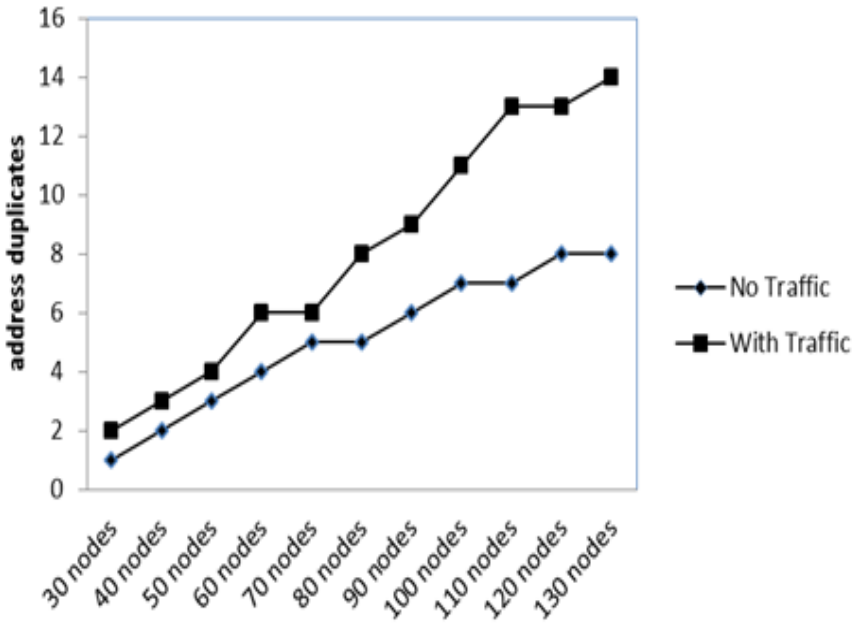

Figure 4. 1: Effect of Network size on address uniqueness

ii. Effect of Network size on communication overhead

Figure 4.2 shows the number of packet transmissions according to the number of nodes. The result shows that the number of packets is in proportion to the number of nodes in both experiments. When traffic was introduced, the number of packets increased slightly. The difference in the communication overhead observed for StrongDAD under the two conditions is not significant. This is due to the fact that DAD is time- based and hence configuration depends mainly on time not the number of packets.

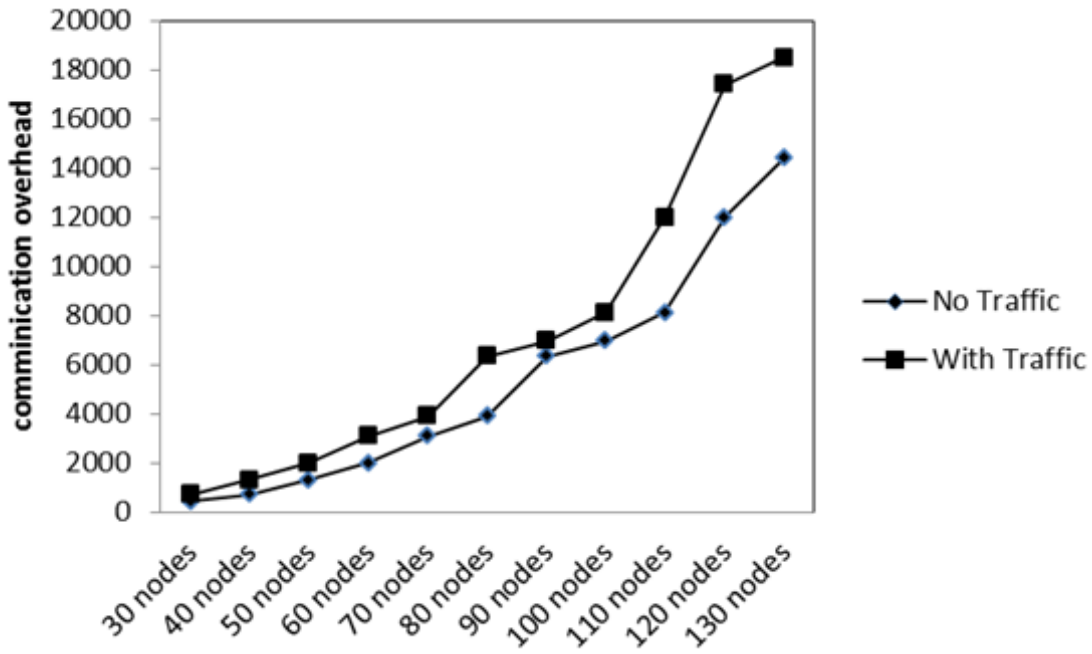

Figure 4. 2: Effect of Network size on communication overhead

iii. Effect of Network size on latency

The results shown in Figure 4.3 illustrate the effect of network size on latency. In both experiments address allocation latency increased proportionally to network size. This was due to the fact that network traffic increased with network size, thereby affecting the delivery of address allocation packets. Increase in traffic caused an increase in latency of address allocation packets. When network traffic was introduced, the latency slightly increased for all network sizes. The DAD timeout period was set at 1.5 seconds and each new node performed two DAD trials. This resulted in not more than 3 seconds latency value for each address allocation operation. Any value above the threshold of 3 seconds contains a delay component incurred while delivering address allocation packets.

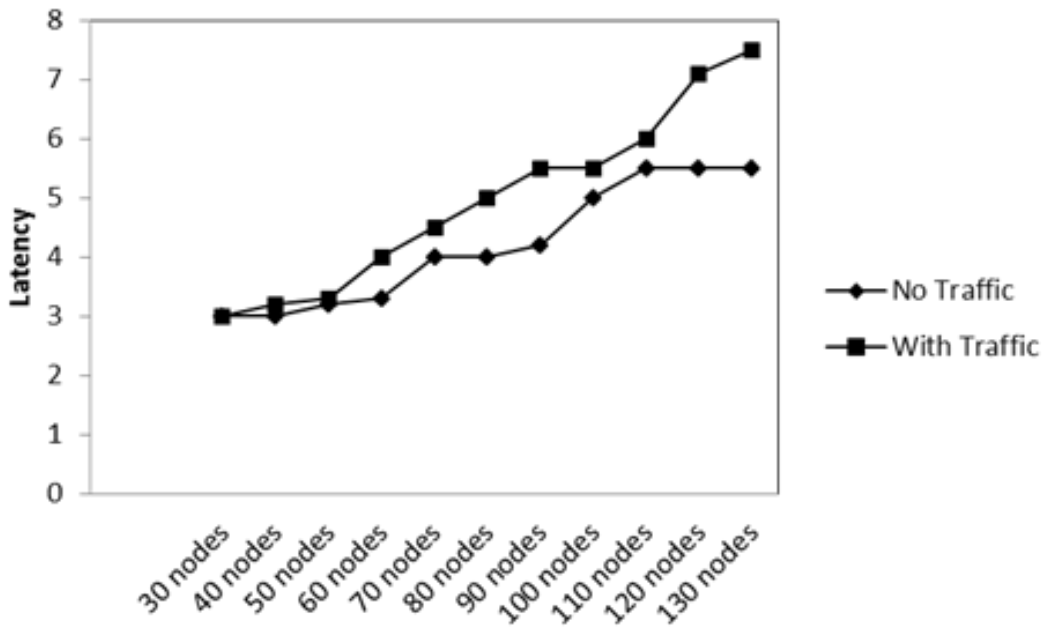

Figure 4. 3: Effect of Network size on latency

b) The Effect of Type of Network Traffic on Address Auto-configuration

This set of experiments investigates the performance of DAD under three traffic models, namely CBR, Pareto, and Exponential. These were generated using the tool cbrgen.tcl. In the experiments, a total of 100 randomly placed and preconfigured nodes were used before new 30 nodes were introduced into the network. The new nodes were introduced at the rate of 1 node every 10 seconds.

CBR Traffic Model: Packets size was set to 64 bytes generated at a constant rate of 2 kb/s. The packet inter-arrival time was set at 600ms. The holding time of the model follows a Pareto distribution with a mean of 300s and a shape parameter of 2.5.

Exponential Traffic Model: During the ON period, the traffic is generated at 2 kb/s. The holding time follows an exponential distribution with a mean of 300s.

Pareto Traffic Model: The ON/OFF periods followed a Pareto distribution, where traffic was generated at 2 kb/s during ON periods. Average ON/OFF periods were 315ms and 325ms respectively. The holding time followed a Pareto distribution with a mean of 300s and a shape parameter of 2.5.

i. Effect of traffic type on address uniqueness

Figure 4.4 shows address duplicates recorded in the four experiments. When the network was subjected to Pareto traffic, address duplicates were slightly higher than in the other three experiments. In Figure 4.5, Pareto recorded lower communication overhead than the other traffic types. Low communication overhead was a result of nodes being able to defend their IP addresses, hence the high address conflicts recorded in Figure 4.4.

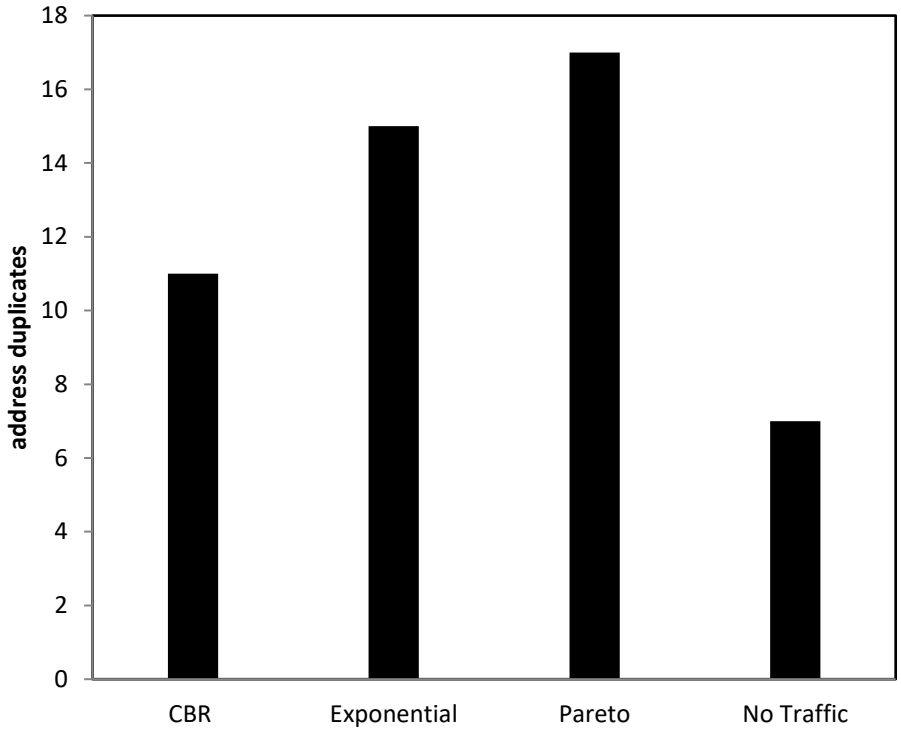

Figure 4. 4: Effect of traffic type on address uniqueness

ii. Effect of traffic type on communication overhead

The amount of communication overhead recorded when the network was subjected to CBR traffic was higher than in the other three experiments. CBR generates traffic at a constant rate. This resulted in more address duplicate notification packets being delivered, hence the address allocation process was started all over again (causing more traffic). On the other hand, the delivery of address conflict messages resulted in far fewer address duplicates for CBR. Parreto, which recorded less communication overhead, yielded more address duplicates than the other three experiments. In other words, fewer address allocation packets might have caused addresses to be duplicated.

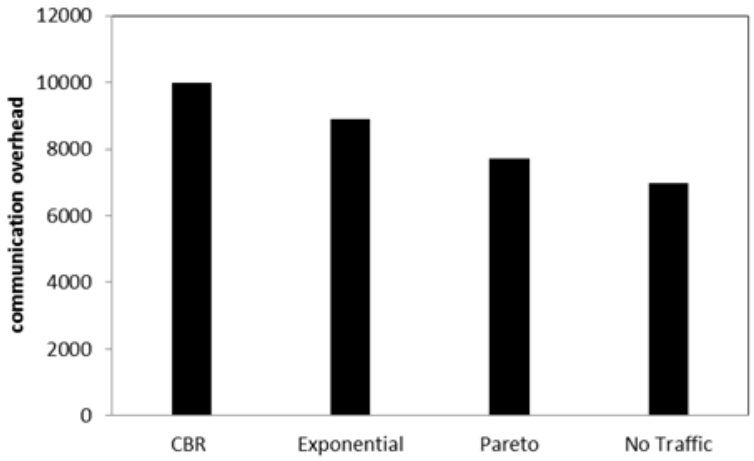

Figure 4. 5: Effect of traffic type on communication overhead

iii. Effect of traffic type on latency

Address allocation latency was not significantly affected by traffic type. The network with exponential traffic recorded slightly more latency that the other three experiments. As shown in Figure 4.6, when no network traffic was present, the latency was slightly lower than in the other three scenarios.

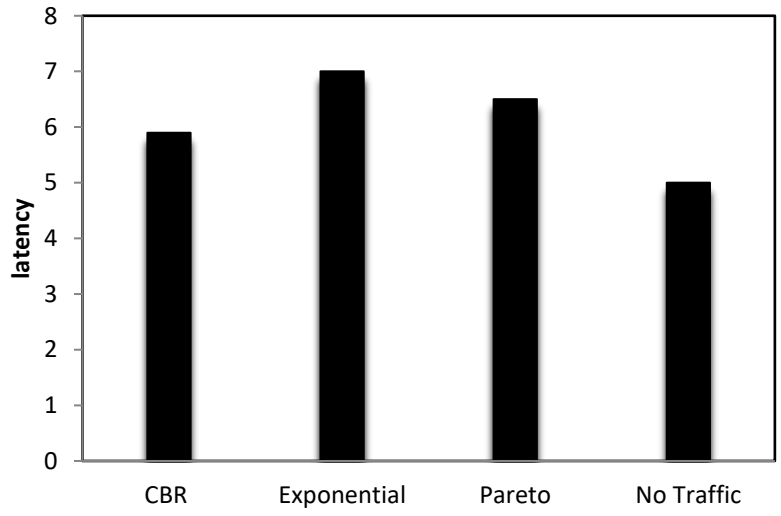

Figure 4. 6: Effect of traffic type on latency

4.2 Experimental Setup: Effect of Node Mobility on Address Auto-configuration

In this sub-section, the effects of node mobility on Address auto-configuration were investigated. The mobility of the nodes affects the number of average connected paths, which in turn affects the delivery of data packets in the network, (Alvarez et al., 2016; Divecha et al.,2007). In the simulation experiments an evaluation of the performance of StrongDAD under the Random Way point Mobility model was done. In the Random Waypoint Model, a node randomly chooses destination coordinates and starts moving towards the destination at a certain speed chosen at random. The speed is chosen from a uniform distribution [0,V_max], where V_max is the maximum speed for every mobile node. After reaching the destination, the node stops for a duration defined by the 'pause time' parameter. After this duration, it again chooses another destination at random and repeats the whole process.

The experiments were conducted in Network Simulator-2 version 3.1 running on the Linux operating system Ubuntu Linux 14.04. CMU extension of ns-2 was part of the implementation to support ad hoc networks. Table 1 below shows the other simulation parameters used in the experiment. The aim of this experiment was to show the effects of mobility on DAD performance using the Random Waypoint Model.

Table 4. 1: Experiment parameters used in Simulation

Parameters	**Environment**
Number of nodes	30, 40, 50, … 150
Area	1000m x 1000m
Simulation time	6000 seconds
Node arrival rate	1 node / 25 seconds
Mobility Model	Random Waypoint mobility model

4.3 Simulation Results: Effect of Mobility on Address Auto-configuration

This section presents the experimental results obtained from the analysis of the effect of node mobility on Address Allocation using StrongDAD.

(a) The Effect of Mobility on Communication overhead

Figure 4.7 presents a graph of the impact mobility has on communication overhead during address auto- configuration. The results showed that as the network size increased

communication overhead also increased. Once mobility was introduced there was a rapid increase in communication overhead, hence the performance of IP auto- configuration protocol was greatly affected. The increase in the overhead resulted in a high rate of lost communication packets as well as an increase in duplicate addresses and conflicts.

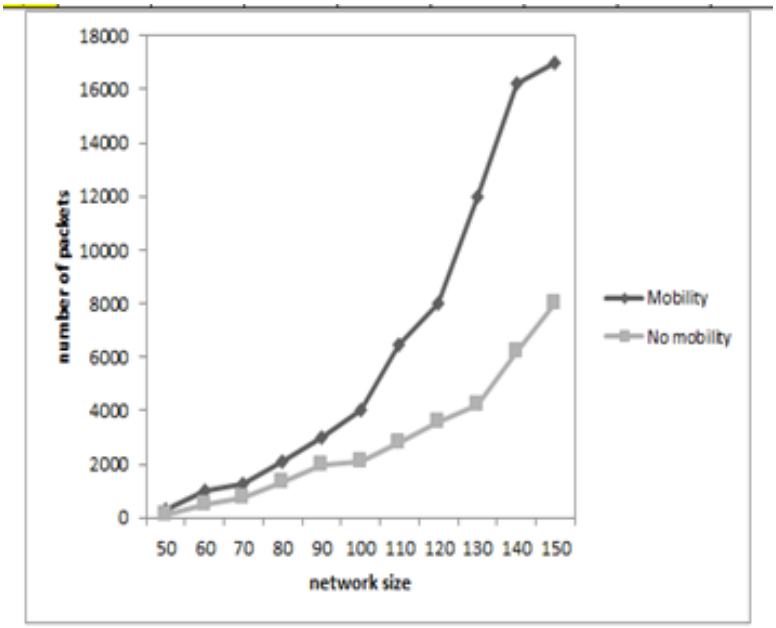

Figure 4. 7: The effect of mobility on communication overhead

(b) Experiment 2: The Effect of Mobility on Address Uniqueness

Figure 4.8 shows the number of address duplicates against the network size. From the results, it was clear that network size had a significant impact on number of address duplicates as they increased rapidly with network size. When mobility was introduced, a much higher number of address duplicates was recorded in all the experiments conducted. Mobility of nodes resulted in more packets being lost, hence some address allocation packets might have been lost. If address allocation packets were lost, then nodes may not

have received address conflict messages, resulting in duplicate addresses. It is recommended to consider mobility when address allocation takes place. Protocols should be able to handle problems emanating from mobility.

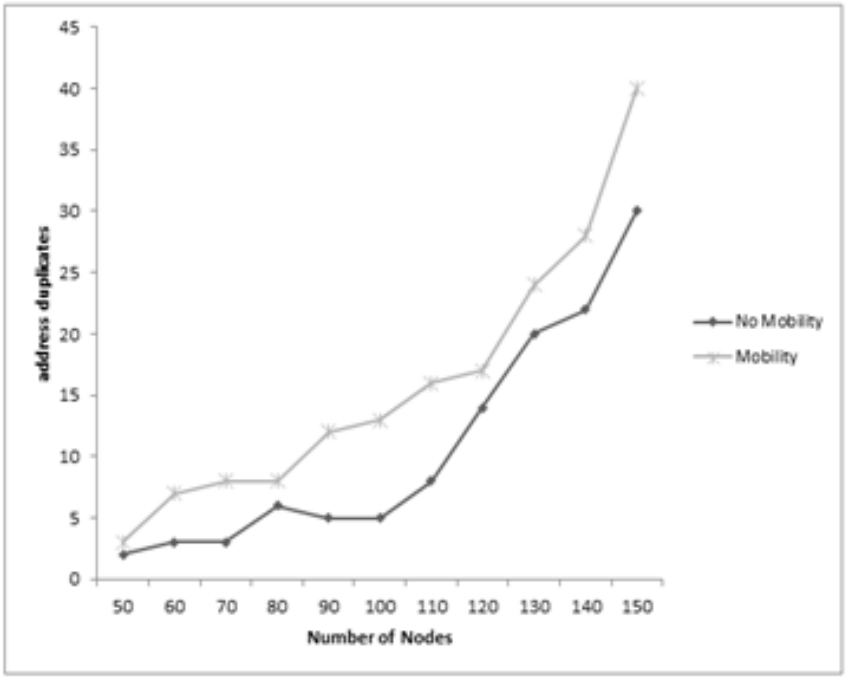

Figure 4. 8: Effect of mobility on address duplicates

(c) Experiment 3: The Effect of Mobility on Latency

The graph in Figure 4.9 shows latency against network size. All experiments show that latency gradually increases with network size. The introduction of mobility gave rise to a rapid increase in latency during address auto- configuration. This pointed to an increase in delays and loss of communication packets in the network. Furthermore, as shown in Figure

4.7, the greater the communication overhead, the greater the loss and delays in the network, thereby increasing latency. This showed that mobility had a negative impact on the effectiveness of any auto- configuration protocol.

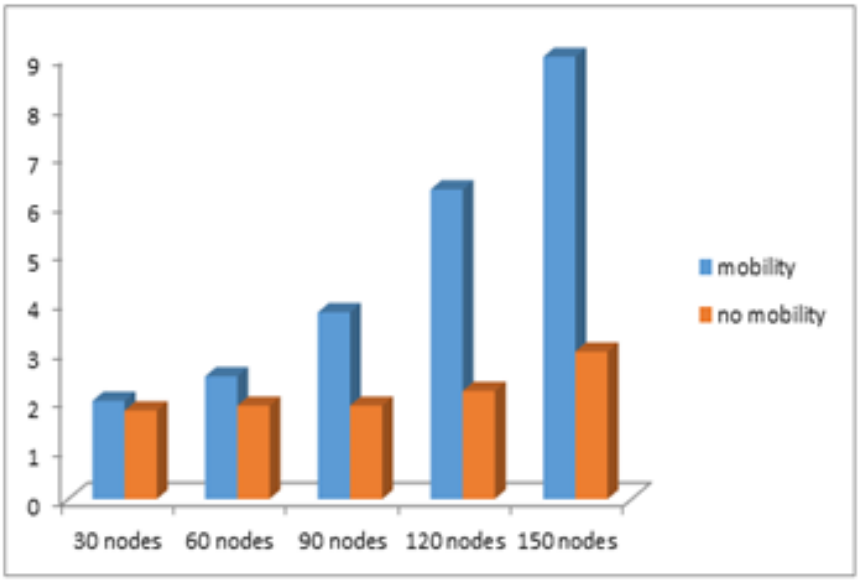

Figure 4.9: Effect of mobility of address latency

4.4 Chapter Summary

Address auto-configuration in wireless ad hoc networks has received a lot of attention in recent years. Many solutions have been proposed and tested. However, little has been done on testing how different network conditions affect the performance of the proposed algorithm. DAD is the cornerstone of many address allocation solutions that follow the

stateless paradigm. This chapter presented experiments to assess the effect of network mobility and traffic on Duplicate Address Detection (DAD) in wireless ad hoc networks. In the experiments, it was observed that the presence of network traffic affects the performance of the address allocation protocol. The introduction of network traffic resulted in the auto-configuration protocol generating more communication overhead and more address conflicts. Latency also increased due to network traffic. This observation calls for further investigation into the improvement of address allocation protocols.

Node Mobility was also found to have different effects on the address auto-configuration mechanism. This is chapter, we considered scenarios where nodes were stationary and compared them with scenarios where mobility was present. Mobility is a key feature for MANETs, hence it is imperative to improve current auto-configuration mechanisms to include adaptive features which can cater for various changes in networks due to node mobility. The effects of mobility call for better design of address auto-configuration protocols. An outline of the design criteria to consider when designing auto-configuration that can work effectively if the nodes are mobile is given below.

Initiator / requestor moving: If a new node requests for an address using an initiator, and the initiator moves before the completion of the address auto-configuration process, this may result in the new node having to wait indefinitely. Address auto-configuration protocols must be able to consider a case where either the requestor or the initiator moves during the address auto-configuration process.

Change in network topology: Nodes in an ad hoc network are highly mobile, meaning that they can leave and join the network at any time, and at any position. This results in rapid changes in network topology and affects the operations of the DAD protocol. Networks can merge or split, thereby affecting the topology and packet delivery, state information updates and the auto configuration process.

Message losses: The address auto-configuration procedure requires that nodes exchange messages. For the address allocation procedure to work properly, all control packets must be delivered to the intended destination. Message delays and losses result in address duplicates. It is therefore imperative to guarantee the delivery of all address allocation packets.

Chapter 5

IP address Auto-Configuration Algorithms for Wireless Ad hoc Networks

5.0 Introduction

This chapter presents an auto-configuration protocol that is based on the theory of swam intelligence that has made rapid progress in the last few years. In-order to provide relevant building blocks for such a solution, further investigations were conducted in chapter 3 and 4. From the results obtained in Chapter 3 we manage to ascertain best setting for DAD timeout period, whilst the results from the experiments conducted in chapter 4 were also used to aid the protocol under any network traffic and node mobility. In the approach proposed in this chapter, we introduce state information maintenance which is passively collected but not actively maintained. In swam systems, this is done to improve cohesion and alignment of the system components. In address auto-configuration, on the other hand, this is done to reduce communication overhead generated by the protocol. In literature, passively collecting state information has been found to reduce the number of DAD trials, thereby reducing latency and communication overhead. Reducing communication overhead inherently conserves bandwidth, thereby improving on QoS of the whole network.

Earlier in this work we identified the need to design protocols that can operate under various network conditions, considering that network conditions will, among other things, help in reducing communication overhead and address duplicates. Reducing communication overhead and improving on address uniqueness will aid higher level protocols to function better, thereby improving QoS provisioning. However, the previous chapters also concluded that it is challenging to build adaptation components without knowing how different network conditions affect the functioning of the address auto-configuration protocols. Chapters 3 and 4 presented experiments to determine the optimal DAD configurations and the effect of network conditions on the address allocation process respectively. The results reported in the two chapters were then used in the design of the auto-configuration algorithms presented in this chapter.

This chapter is structured as follows. Section 5.1 outlines the design system architecture of the proposed protocol. A detailed design of this protocol is covered in sections 5.2 to section 5.6. Section 5.7 concludes the chapter.

5.1 The Dynamic DAD Address Allocation Protocol – System Architecture

In this section we present D-DAD, an adaptive address allocation protocol. This protocol is based on the model earlier formulated in chapter 2 and also on the investigations carried out in chapters 3 and 4. In these chapters we investigated how network parameters such as traffic affect address allocation protocols. StrongDAD was used as a test case because the protocol proposed in this chapter is based on StrongDAD. The investigation carried out in chapter 3 concluded that the value of DAD timeout period should be between 1 second and 1.4 seconds and must be decided at runtime, contrary to existing proposals in the literature. Furthermore, the investigation presented in chapter 4 reveals the importance of considering the amount and type of traffic flowing in the network when soliciting for an IP address. In this work, we harness all this knowledge and design an IP address auto-configuration protocol that is more robust.

The D-DAD protocol is based on the following three key guidelines proposed in chapter 2. (From these characteristics, we developed the building blocks of the proposed algorithms. The general architecture of the proposed algorithm is given in Figure 5.1 below).

(a) Addresses should be distributed to all the nodes: To distribute addresses, nodes passively collect information about their neighborhood. This information is propagated from protocol control messages. No central management of IP addresses is used. Any node in the network should be able to allocate IP addresses to new nodes.

(b) State information must not be explicitly synchronised: State information is propagated using routing protocol packets and hello messages. No network- wide broadcast is used. Nodes passively collect state information packets. This method is adopted from work described in Wise-DAD (Mutanga et al., 2008)

(c) The protocol should adapt to changes: To adapt to network changes, two components are proposed, namely, the monitoring and adaptation mechanisms. The solution has algorithms that monitor the following:

i. network merging & partitioning
ii. network size
iii. network traffic volume
iv. mobility

Each of the above network conditions is passively monitored by the protocol. Each condition calls for different adjustments to the auto-configuration procedure. In the sub-sections below, details of how the proposed protocol configures IP addresses, adapt to changes and monitors network accordingly are given.

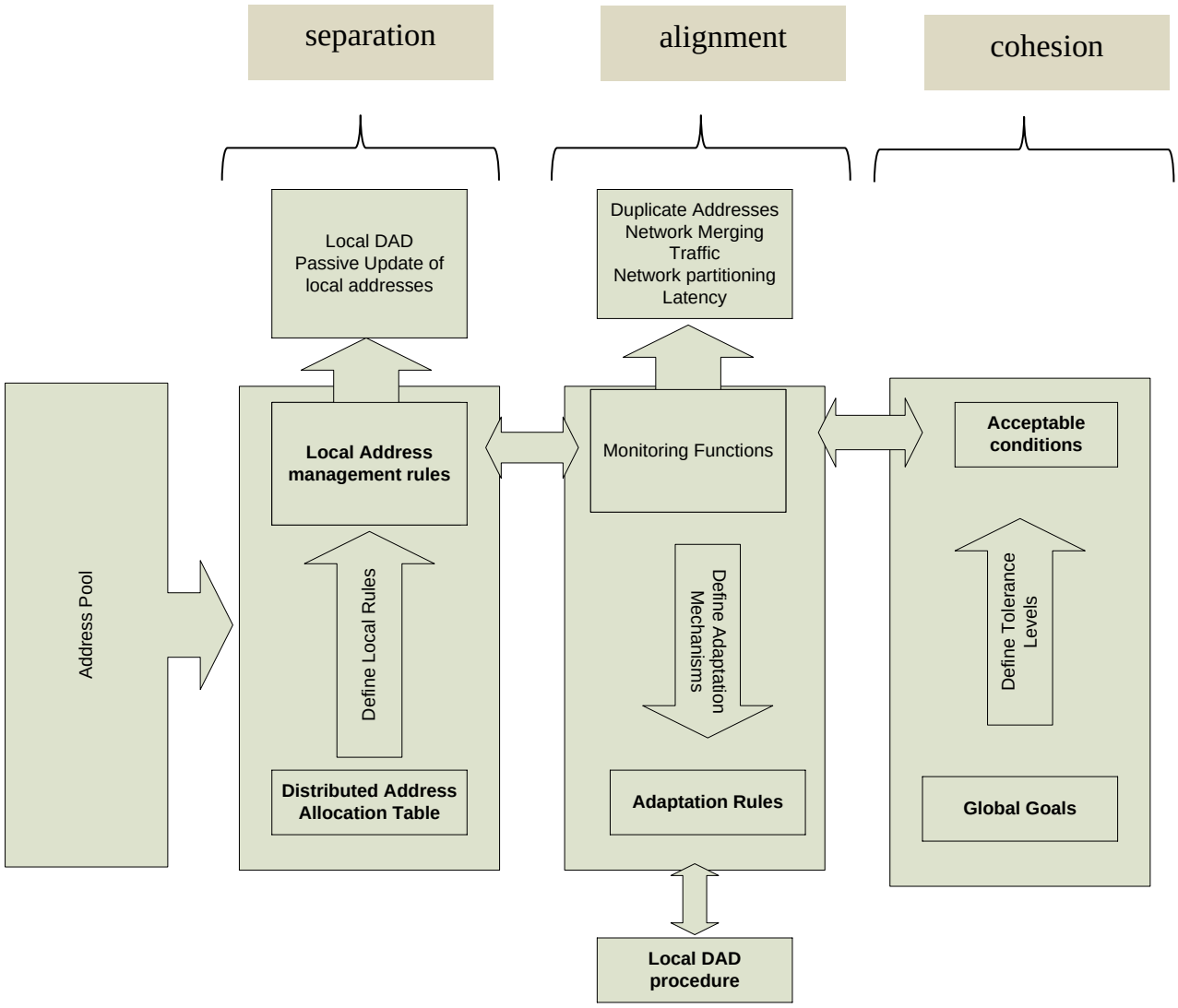

Figure 5. 1 D-DAD address auto-configuration protocol

The problem of IP address auto-configuration and maintenance can be divided according to the following functions:

(a) Network formation*:* This function is responsible for the formation of the network. It also handles how the nodes decide who chooses network parameters such as the network identifier, DAD timeout period, and the number of DAD trials.

(b) Node admission*:* This function deals with how nodes join the network, i.e. how new nodes acquire IP addresses. The problem of how the current membership decide what parameters to use when a new node joins the network is also addressed.

(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 responsible for handling the problem of network merging.

(e) ***Network partitioning:*** This function handles problems that arise as a result of network partitioning.

5.2 Network formation

A node that is not configured and wishes to join the network periodically broadcasts a '***request to join'*** message and sets a timer (Join Timer). The ***request to join*** message contains the node's temporary Host Identity number (HID). The message is sent only to one- hop neighbours and cannot be rebroadcast. When the Join timer expires, the node will rebroadcast the message and reset the timer again until at least one of the immediate neighbouring nodes replies. A network comes into existence if an unconfigured node receives a request to join from another unconfigured node.

Upon receiving a request to join message, an unconfigured node checks the HID encapsulated in the message and compares it with its own. The node with the lowest HID becomes a temporary leader and proceeds by choosing the Network identifier (NID) and sends it to the other node. The leader also sends an IP address that it generates at random from a range of valid IP addresses for the network. The second node will also choose its own IP address and notify the other node of its chosen IP address. From that point onwards, a network of two nodes starts to exist. New nodes can then join the network by sending a request to join the network to any of the two configured nodes.

```
method receive-Packet()
Begin
        if (packet_type = REQUEST_TO_JOIN) then
                Begin
                        If (configured = FALSE) then
                        Begin
                                If (this.HID <message_HID) then
                                Begin
                                    GenerateNID();
                                    Send-Message(AddressReply);
                                End
                        End
                         Else If (configured = TRUE) then
                        Send-Message(confirmation);
                End
      Else call appropriate method;
End
```

Figure 5.2: Processing a request to join message

5.3 Node Admission

A configured node receiving a request to join message replies with a confirmation message to the sender. The confirmation message signifies that the configured node can act as an initiator for the new node. 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*** as a way of indicating the chosen initiator. This is a way of making sure that only one node can act as an initiator for a new node. A configured node cannot act as an initiator for more than one requestor.

The initiators send network parameters that allow the new node to generate its temporary address and request for a permanent address. The network parameters include: (1) average node density; (2) network size; (3) traffic type and volume; (4) merging and partitioning status of the network; (5) Address allocation table. This information is used by the new node when sending a request for an IP address. The new node or the initiator generates a random IP address and checks if it is in the allocation table received from an initiator 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 (DAD timeout). The duration of the DAD timeout period depends on the network conditions. According to the earlier investigation presented in chapter 3, this value of DAD timeout period is the maximum time. Any other duration longer than that will result in unnecessary latency whilst anything shorter may result in address conflicts.

If the broadcast timer expires (after the calculated DAD value) without any node defending the requested IP address, the requested address is assumed to be free. The initiator completes the address allocation process sending 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 to them. A message sequence number is used to determine if a message is new or not. We adopted the use of message sequence numbers from routing protocols such as DSDV. If the requested

address is found to be in use, an IP conflict message is sent to the initiator and the process is repeated. If the message is not new, it is discarded, otherwise it will be rebroadcast until it reaches all network nodes.

Before the message is rebroadcast, the recipient appends its own 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 conserves bandwidth by enabling the IP conflict message to be unicast back to the initiator. When nodes receive an AREQ, they also update their allocation tables using IP addresses in the reverse path list before rebroadcasting the AREQ. This allows for passively collecting state information without adding communication overhead.

```
Begin
        If (packet_type NOT AREQ) then
        Call appropriate method
        Else
                Begin
                        If (messageNEW= FALSE) then
                        Drop-Packet
                        Else
                                if (this.IP-Address  != RequestedIP)
                                Begin
                                        AppendIP-Address()
                                        UpdateAllocationTable()
                                        UpdateMessage-Segquence-Number ()
                                        If this.NodeStatus = Mobile
                                        Begin
                                                AppendMobility Status
                                        End
                                        Send-Message(AREQ)
                                End
                                If (this.IP-Address == RequestedIP)
                                Begin
                                        Send-Message(ConflictMessage);
                                End
                End
End
```

Figure 5.3: Processing Address Request packet

(a) Handling mobile Nodes: Propagation of the IP conflict message

One network condition that was considered in this work is mobility. During the auto-configuration process, some vital nodes may move, hence affecting the delivery of address auto-configuration packets. Earlier, in chapter 4, an investigation of the effect of mobility on the address auto-configuration protocol was conducted. The investigation revealed that IP address conflict messages are lost due to mobility of the nodes involved in the address assignment process. The impact of mobility was reported in detail in Chapter 4. In this sub-section, a solution to that problem is proposed. There are two cases that were considered in this thesis:

i. Mobile requestor or initiator: In this work, we propose that, when requesting for an IP address, a node should indicate its mobility status. This message is passed on to the initiator, which subsequently passes it to the rest on the network during the DAD process. If a requestor's status is recorded as a mobile node, the IP address conflict is not sent in unicast. The message is broadcast in the same fashion as the IP address request message is propagated in the network. This process guarantees the successful delivery of an IP address conflict message even when nodes are mobile. In this case, the address auto-configuration does not use an initiator. The initiator only sends network parameters that the new node uses to perform DAD.
ii. Mobile intermediate node: During the address solicitation process, an IP address request message may pass via a mobile node. Instead of just appending its IP address

to the message, the intermediate node flags the message with mobility status information. If a node wants to defend its IP address, it chooses the path that does not have a mobile node. This is possible because an IP request message may take different routes to reach its destination. If all routes have at least one mobile node, the IP conflict message is broadcast instead of unicast.

(b) Handling network traffic volume or traffic type

The volume and type of network traffic have an effect on the performance of the address auto-configuration protocol. In the previous chapter, we investigated the impact that network traffic has on the address allocation process and concluded that there is a need to take network traffic into account when designing address allocation solutions.

Larger and more exponential volumes of traffic affect the address auto-configuration process adversely, hence the design of address auto-configuration protocols must take this into account. In the experiments reported in the previous chapter, we noted that IP address conflicts sent during the configuration procedure were lost. To minimise the effect of the message losses, we propose that the number of DAD trials be increased. By default, a new node performs DAD only once but may perform it twice if the volume of traffic reaches a certain threshold. However, we did not model traffic volumes but simply recorded the traffic volumes from both forwarded and sent packets.

```
method receive-Packet
Begin
      If (packet_type = confirmation && MobileStatus == 0) then
       Begin


                        While Timer NOT Expired
                        Begin
                                    if initiator mobile.status  = 1
              Begin
                                    StoreInitiator;
                                    Receive-Packet()
                                    End
                        End
                        If All Initiators = Mobile
                        Begin
                                    Choose First Initiator;
                                    DAD_Status = Broadcast;
                                    DAD_Trails  = 1;
                                    GenerateIP()
                                    Send-Packet(AREQ)
                        End
                        Else
                                    Begin
                                                Choose Stationary Initiator
                                                Send-Packet(initiator-selection)
                                    End

                        If network  = High
                        DAD_Trials = 2 ;
                        Else
                        DAD_Trials  = 1;


      End
      Else

              If (MobileStatus  == 1)
              Begin
                        DAD_Status = Broadcast;
                        DAD_Trails  = 1;
                        GenerateIP()
                        Send-Packet(AREQ)

              End

End
```

Figure 5.4: Unconfigured node processing a confirmation packet

5.4 Node departure

Node departures can either be graceful or abrupt. With graceful node departures, the node has time to shut down and inform its network peers, whereas abrupt departures may be caused by factors such as mobility and power problems. If a node departs gracefully, it notifies its peers by broadcasting a goodbye message.

All nodes that receive the goodbye message erase the departing node's IP address from their address allocation tables. This procedure enables the re-use of the addresses previously allocated to old nodes. Before a goodbye message is processed nodes first check whether the message is new or not. If it is old, it is discarded otherwise it will be broadcast. When nodes receive a goodbye message, they also update their state information using the IP address allocation table contained within the data packet.

```
method receive-Packet
Begin
    If(packet_type = Departure)then
    Begin
        If(Departure_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 5.5: Processing of goodbye message

In the case of an abrupt node departure, the node does not have time to inform its peers 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 eventually deleted from all 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. If the node is still in the network it responds to address requests by sending a conflict message. This is to guard against allocating an address that is still in use and thereby causing address conflicts.

5.5 Detection of Network Partitions

In the proposed approach, nodes monitor not only their neighborhood but the whole network for signs of network partitioning. This thesis argues that monitoring the neighborhood alone is not sufficient to detect a network partition. Mobility and other related issues may cause a sudden change in neighborhood without necessarily causing network partitioning. Temporary disconnection must not be treated as network partitioning, hence the need to distinguish between the two. The challenge, therefore, is how can nodes monitor the whole network without compromising on the bandwidth?

In this approach, network merging is detected by a set of nodes, **K**. All nodes in the network are connected directly or indirectly to at least one node in the set **K**. Any node should be, at most, two hops away from at least one node in **K**. In other words, a node should either

be connected to a node in set **K** or one of its neighbours should have a direct connection with any node in **K**. To make sure that each node is connected to **K**, when nodes join the network, they check if any member of **K** exists in their neighborhood. If not, they make themselves part of **K** and update the whole network by broadcasting a message. This method ensures that the members of **K** are evenly distributed in the network.

Network partitioning is detected if a certain number of nodes in **K** are no longer reachable. Nodes constantly check for any changes in the membership of **K**. If only a small portion of K is not reachable the network does not change its network ID but rather puts all nodes on high alert. On high alert, nodes constantly scan for the missing part of the network and, if the time **W** has not expired, the nodes do not allocate addresses that belong to the other network. If the threshold **T** (part of **K**) is missing, the network partition is classified as temporary until a time period **W** has passed. If the time **W** has not expired, the nodes do not allocate addresses that belong to the other network. This is to allow for the two networks to seamlessly merge at some time. If after a certain period **W** the network has not been restored the network is then classified as being partitioned and a new network ID is generated. For the purposes of simulation, the value of **W** was set to 500 seconds. Also, for the purposes of the simulations, the value of the threshold **T** was set as half of set **K**. It is, however, imperative that an optimum value for **W** and **T** be established. Network dynamics and usage scenarios of the network may affect the choice of these values.

5.6 Detecting and managing network merging

Two cases of merging networks are considered in this work. First, we consider merging of networks that were previously one network. After network partition, the networks may have classified the partitioning as either temporary or permanent. If the partitioning was classified as permanent the protocol treats the network merger as if the two were independently configured. On the other hand, if the network partitioning was classified as temporary, the protocol handles this occurrence differently.

If the two networks have been part of the same network, their network ID will be the same. As described above, if a network partitions into two, the partitioning is regarded as temporary until a certain time **W** has expired. If the two networks merge again before the expiry of time **W**, network merging can take place without change of IP addresses. This is possible because, if the time **W** has not expired, the nodes do not allocate addresses that belong to the other network, hence there is no need for address changes after the two networks come into contact. Once the two networks merge, all nodes remove the status of temporary partition from their entries. The set **K** is updated and the information is propagated to all the network nodes using a broadcast message.

Independently configured networks may come into each other's transmission range and network merging can occur. We assume that the address allocation table is distributed to all the nodes. We also assume the existence of a mechanism to synchronise the tables

periodically. This, however, is not part of the network merging solution but part of the initial configuration mechanisms for new nodes. The algorithms that handle both the distribution and synchronisation of IP address allocation tables are explained in the previous sections.

Once network merging has been detected, IP address conflicts are detected by exchanging the address allocation tables. The merging networks exchange their address allocation tables using a network- wide broadcast. The node that detects the network merging initiates this process.

If there are two nodes with the same IP address, the nodes that come from a network with fewer nodes relinquish their address and acquire a new one. From the address allocation tables an estimate of the network size of each of the merging networks can be obtained. Nodes can only change their network IDs once the IP address conflicts are resolved. The nodes whose addresses are not affected by the network merging change only their network IDs.

5.7 Chapter Summary

This chapter presented an address auto-configuration protocol based on the model presented in Chapter 2. The auto-configuration protocol proposed in this chapter adapts to node mobility, traffic and network size. We argue that the unpredictable nature of the wireless ad hoc networking environment presents a number of challenges. D-DAD follows the stateless address auto-configuration paradigm with a passively synchronised address allocation to reduce the number of DAD trials. The protocol proposed in this thesis consists of four main components, namely, network formation, node admission, network merging, and node departure. The following chapter evaluates the proposed algorithms through simulation experiments conducted in the NS2 simulator.

Chapter 6

Performance Evaluation of the D-DAD protocol

6.0 Introduction

In the previous chapter, we presented the D-DAD IP address auto-configuration protocol. which is based on the adaptive model proposed in Chapter 2. As a way of testing the building components of the proposed model, this chapter presents an evaluation of the proposed protocol. In this chapter, we demonstrate the effectiveness of the swam intelligent based paradigm in improving the effectiveness of address auto-configuration.

Due to the building components of swam systems D-DAD was engineered around the stateless paradigm with global state upkeep which reduced the number of DAD trials. The experiments considered a wireless ad hoc network with no association with the outside world like the Internet. It was also assumed that the range of valid IP addresses used in the network is known ahead of time. Only for the purpose of delineation, we considered the network to be a private IP Version 4 network structure capable of using either 8 bits or 16 bits for node addresses whilst the rest of the bits are held for the network identifier. In any case, the proposed model is just as pertinent to networks utilising the IPv6 address space.

The simulation environment and parameters are described in section 6.2. Various simulation parameters such as the number of nodes, node arrival rate, simulation area, and node density were varied in order to gain a comprehensive analysis. In Section 6.3 we present the experiments performed and an analysis of the results that were obtained. Each experiment was performed ten times and the average values were used for analysis. We compared the proposed protocol against the Wise-DAD (Mutanga et al., 2008) with StrongDAD protocol (Perkins et al., 2001). In Section 6.4 we present the conclusion of this chapter. The architectural details of both StrongDAD and Wise-DAD are given in chapter 2. StrongDAD is a purely time-based stateless auto-configuration protocol whilst the Wise-DAD protocol utilises DAD with a passively synchronised address allocation table. The proposed protocol, D-DAD, is based on the notion of StrongDAD whilst it adopts the use of passively synchronised state information such as the one proposed in Wise-DAD. It is for this reason that D-DAD is compared with the two protocols.

The three protocols were simulated in version 2.31 of the Network Simulator-2 tool running on the Ubuntu Linux 14.04 operating system with CMU extension of ns-2 to support ad hoc networks. Like the previous experiments in chapters 3 and 4, the following metrics were chosen to evaluate the relative performances of all three protocols: Latency, Communication Overhead, Address conflicts.

In section 6.1 we present experiments conducted to determine the effect of network size on the proposed protocol in order to test the scalability of the proposed protocol. Section 6.2 establishes the ability of the proposed protocol to handle high rates of node arrival, whilst

the effect of node density on D-DAD is investigated in experiments presented in section 6.3. Results obtained show that the swam inspired protocol is able maintain the desired global goal by adapting and re-organising when network conditions change. Address conflicts and latency was not adversely affected by drastic changes in network conditions. Section 6.4 presents the results of the investigation of the effect of network traffic and volume on the D-DAD protocol, whilst section 6.5 presents experiments conducted to investigate the effect of node mobility on the proposed protocol. From section 6.6 to section 6.9, results on the performance of the network merging and partitioning algorithms are presented.

6.1 Experimental Setup

D-DAD was simulated in version 2.31 of the Network Simulator-2 tool running on Ubuntu Linux 17.04 operating system with CMU extension of ns-2 to support ad-hoc networks. Figure 6.1 gives a diagrammatic representation of the simulation model. NS2 consists of two key languages: C++ and Object-oriented Tool Command Language (OTcl). While the C++ defines the internal mechanism (i.e., a backend) of the simulation objects, the OTcl sets up simulation by assembling and configuring the objects as well as scheduling discrete events. The C++ and the OTcl are linked together using TclCL. The D-DAD protocol was implemented in C++ and OTcl scripts to test the protocol were developed in the upper layer. OTcl scripts enabled us to create different network conditions to test the protocol.

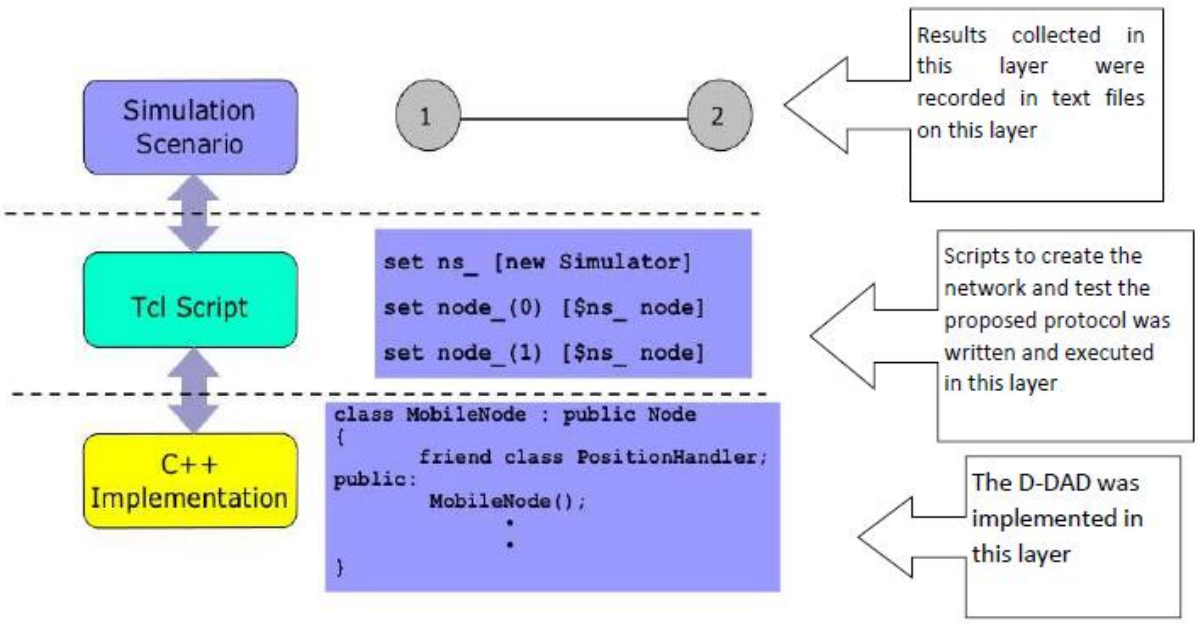

Figure 6.1 : Simulation model

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

vi. Routing Protocol

Nodes were configured to use the Dynamic Source Routing (DSR) protocol. However, the protocols is 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. D-DAD has 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.

vii. 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.

viii. 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.

ix. 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.

6.2 Effect of network size on D-DAD Protocol

The purpose of this experiment was to test the scalability of the proposed protocol by investigating its performance as the network size increases. At the beginning of the simulation, we configured a single node to allow for the other nodes to join. The coordinates of the nodes joining the network were randomly generated by the simulator. We simulated scenarios where all nodes were reachable. This meant that every node had a connection directly or otherwise with the others during the entire duration of the simulation. Node departures due to node failure or mobility were not simulated since the aim of the experiment was to investigate the performance of the proposed protocol as we

increased the network size. Table 6.1 shows the other simulation parameters for the experiment.

Table 6. 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
Observed parameters	Latency, number of received packets, number of address conflicts
Address range	256

i. Effect of network size on communication overhead

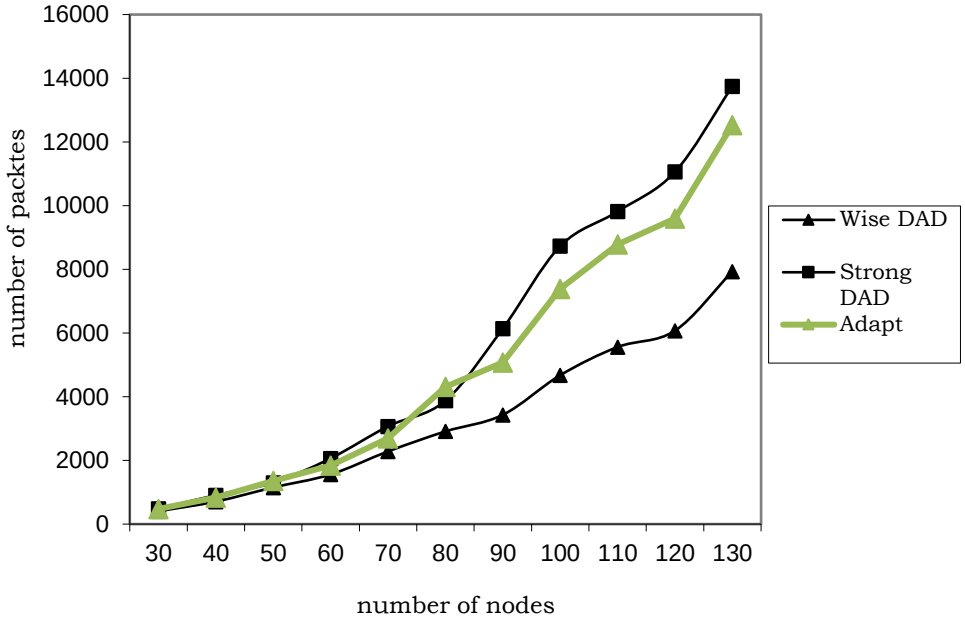

Figure 6. 1: Communication overhead

The amount of communication overhead increases proportionally with network size due to the broadcast nature of all the auto-configuration protocols under consideration. As depicted in Figure 6.1, Wise-DAD had the least amount of communication overhead whilst StrongDAD had the most communication overhead. This is due to the fact that StrongDAD performs a network- wide flooding procedure three times before an address can be assigned. On the other hand, both Wise-DAD and D-DAD perform network wide flooding only once on smaller network sizes.

D-DAD had more communication overhead because, as the network size increases, D-DAD tends to increase the number of network-wide flooding procedures. In addition, if any message losses are detected, D-DAD adapts by performing more network- wide flooding of address configuration packets. The increase in network size is likely to have led to the two conditions, hence the increase in communication overhead for D-DAD. Interference significantly affected communication overhead recorded in StrongDAD. 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 (Figure 6.3)

The increase of communication overhead during automatic configuration will always be proportional to the size of the network. Although this thesis managed to significantly achieve other goals such as reduction of address conflicts, the issues of communication overhead remain a challenge. As more services and protocols are deployed on the wireless channel, it is evident that the need for high capacity channels is of paramount importance. This thesis therefore aurgues that design of hardware for wireless channels

ii. Effect of network size on latency

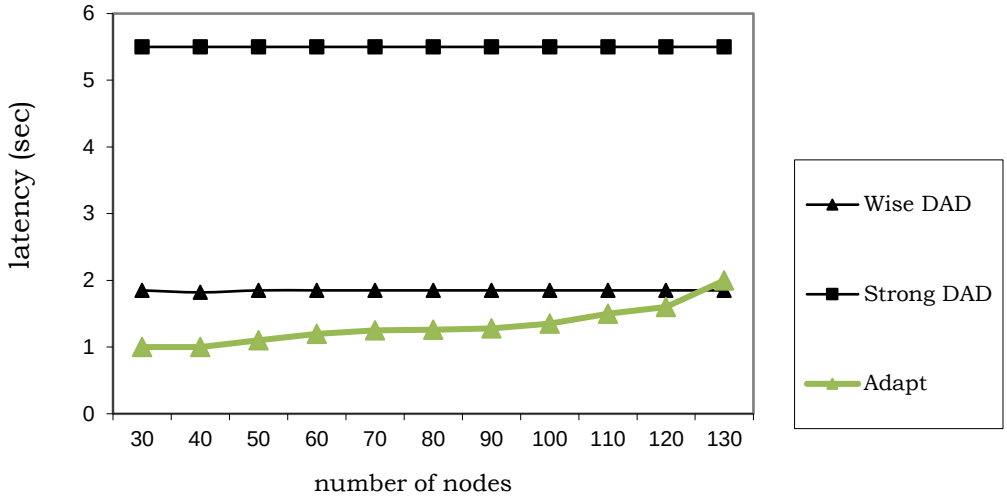

Figure 6. 2: Latency against network size

In DAD-based protocols, address allocation latency is partially affected by the length of DAD timeout period and the number of DAD trials performed before an address can be allocated. For Wise-DAD, the default setting for DAD timeout is a static 1.8 seconds with only one DAD trial. On the other hand, the minimum of 1 second and a maximum of 1.4 seconds was employed in D-DAD. The exact time for the timeout period is determined at run time depending on the conditions of the network. D-DAD adjusts the DAD trials to two as the network size increases and also adjusts the length of the DAD timeout period as network traffic increases. In the experiments presented in Figure 6.2, no traffic was simulated hence D-DAD only adjusted the DAD trials. In StrongDAD, the timeout for

address request (DAD procedure) is 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). These setting contributed significantly to the results obtained. Because StrongDAD performed the procedure three times, the total latency was at least 5.4 seconds.

iii. Effect of network size on address uniqueness

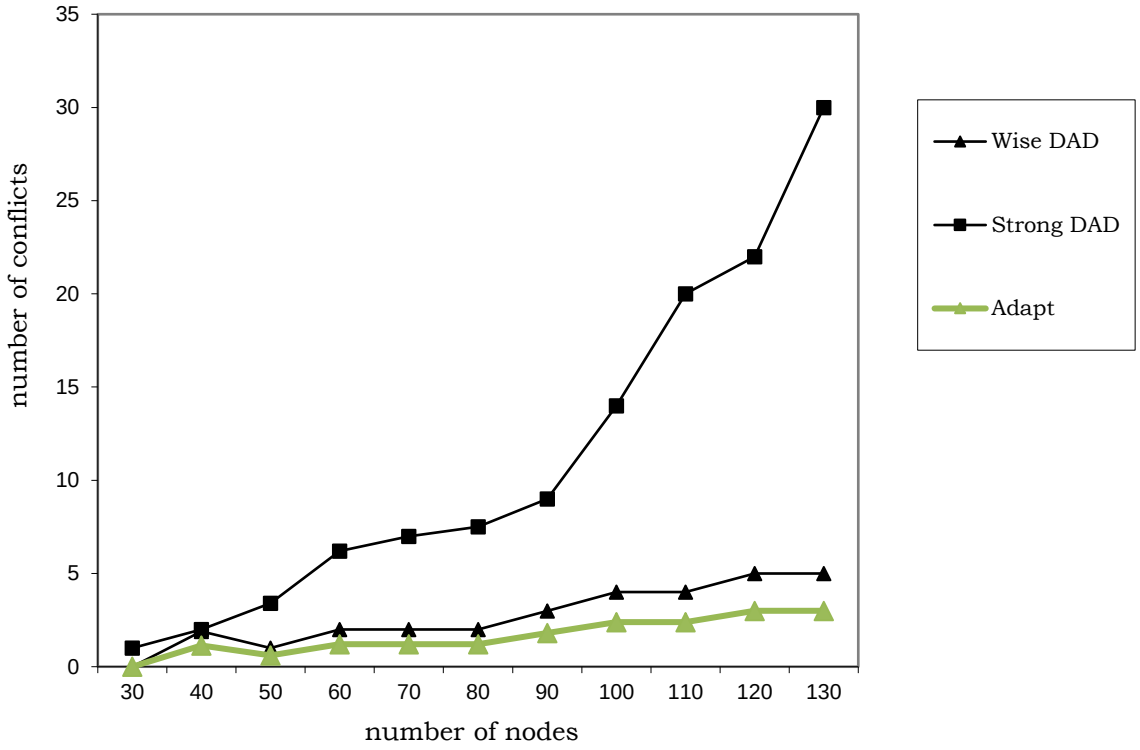

Figure 6. 3: Address duplicates against network size

An important requirement for address allocation protocols is that the configured addresses should be unique. The address allocation protocols should guarantee the uniqueness of the allocated addresses. Address duplication may occur due to erroneous allocation or network merging. Address conflict resolution due to network merging is dealt with in section 6.6. In figure 6.3 we show the number of conflicting IP addresses against varying network sizes. The number of IP address conflicts in the three protocols increases as the number of nodes increases. This is a result of the fact that address space is a finite domain, hence the probability of getting a free IP address decreases as network size increases. Another reason is message losses caused by Medium Access Control (MAC) collisions as network traffic and interference increase. StrongDAD recorded a significantly high number of address conflicts due to the fact that it generated more communication overhead hence more message. Address allocation tables maintained by both Wise-DAD and D-DAD contributed to the significantly low number of address conflicts.

Although low, the number of address conflicts achieved by D-DAD is still unacceptable. The problems emanating from address duplicates are so destructive hence there shouldn't be any compromise when it comes to address conflicts. DHCP based wired networks employ mechanisms to isolate nodes that have duplicate addresses until they are resolved. Going forward, such mechanisms should be considered for wireless ad-hoc networks. Probably the use of a central node or clusters can be considered as a solution. The distribution of the address allocation table seems not to be effective in addressing this problem.

6.3 Effect of node arrival rate on Address Auto-configuration

To test the robustness of the proposed address allocation mechanism we varied arrival rate of the nodes. The network size was fixed at 80 (the median of experiment I). Like the previous experiment, address conflicts, communication overhead, and address allocation latency for D-DAD, StrongDAD and Wise-DAD were recorded for analysis. For the entire duration of the simulation 100% network connectivity was maintained. No nodes were allowed to leave the network. Table 6.2 shows the other simulation parameters used in the experiment.

Table 6. 2 : Simulation parameters for experiment II

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

i. Effect of Node arrival rate on communication overhead

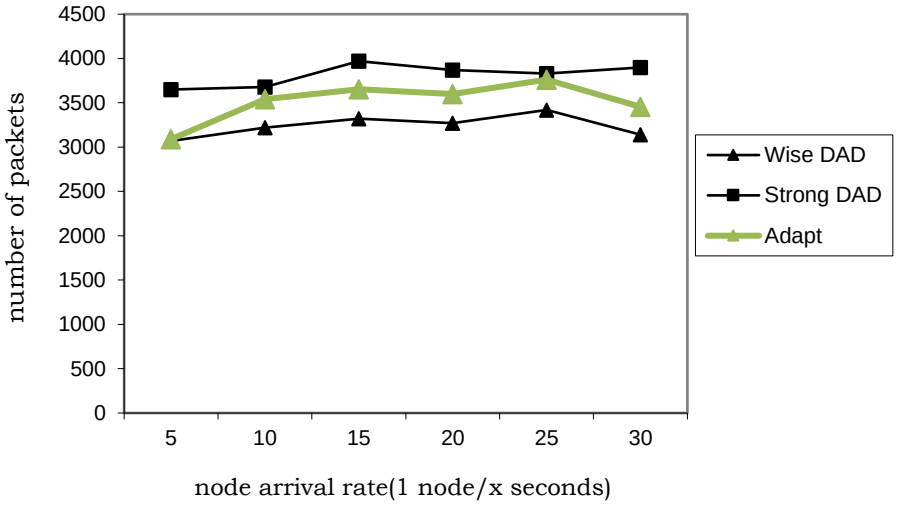

Figure 6. 4: Communication overhead against node arrival rate

Figure 6.4 above shows that the amount of communication overhead generated by the three protocols is not affected by the rate at which nodes join the network. StrongDAD recorded more communication overhead than Wise-DAD and D-DAD. Wise-DAD had the least amount of communication overhead. The minor variations in the number of packets recorded for each of the three protocols are not sufficient to suggest that node arrival rate has an effect on the communication overhead. This is due to the fact that the number of address assignment packets sent by an initiator during the auto-configuration process depends only on the success of a DAD process. Node arrival rate has no effect on DAD success or failure, hence there were no significant variations in the communication overhead as the rate at which nodes joined the network was varied.

ii. Node arrival rate on latency

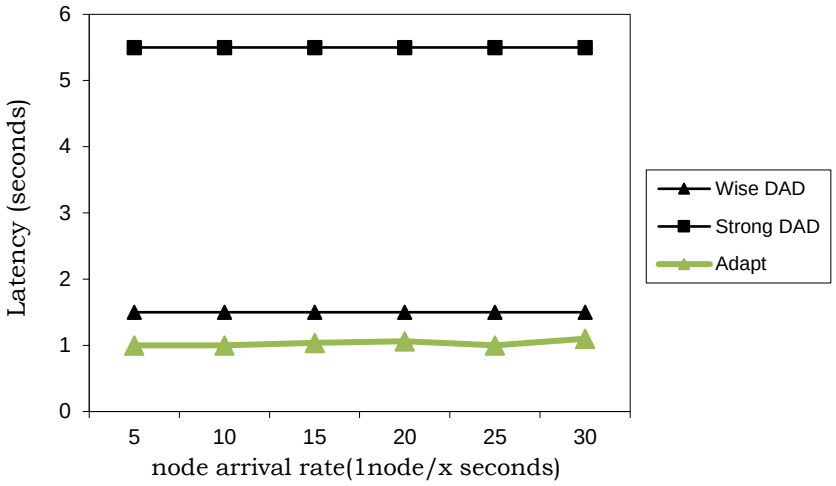

Figure 6. 5: Latency against node arrival rate

In stateless protocols, the address allocation latency is directly proportional to the length of the DAD timeout period and the number of DAD trials performed. The fact that the three protocols performed DAD with different settings of DAD timeout period is reflected in the results shown in Figure 6.5 above. The rate at which the nodes joined the network does not have a bearing on the amount of time taken to configure addresses.

D-DAD had the least latency because its DAD timeout period was set at 1 second. On the other hand, Wise-DAD had a default static setting of 1.4 seconds and performed DAD only once, hence the observed result.

StrongDAD recorded the highest latency of at least 5.4 seconds because the DAD procedure was performed three time with a timeout period of 1.4 seconds for each DAD trial.

iii. Effect of node arrival rate on address uniqueness

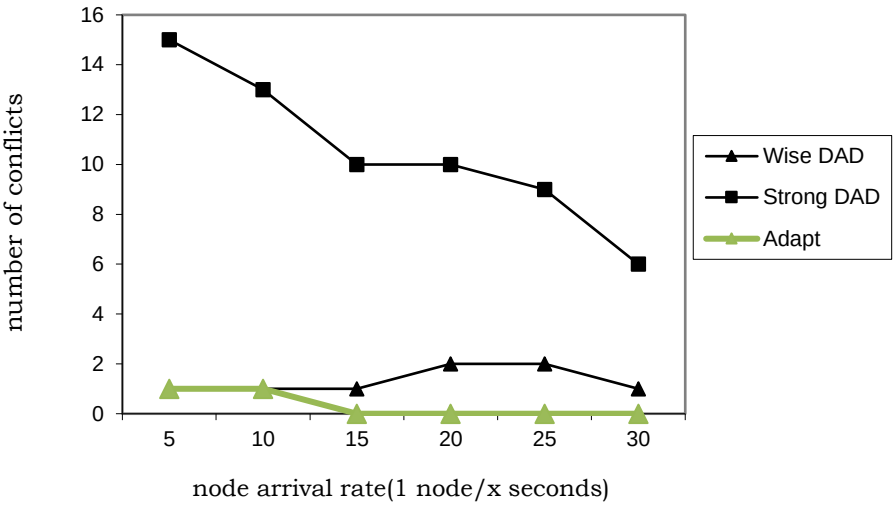

Figure 6. 6 Address duplicates against node arrival rate

Figure 6.6 shows that both D-DAD and Wise-DAD did not show a significant change in the number of IP address conflicts as node arrival rate was varied.

This is due to the fact that state information is updated as new nodes are admitted. The more accurate the state information, the less likely that address conflicts would occur. In a more dynamic network where network membership is highly unpredictable, it is important that the protocol employ an active duplicate address detection mechanism.

On the other hand, there was a significant decrease in address conflicts when StrongDAD was subjected to a lower rate of node admission. A high rate of nodes joining resulted in

more address duplicates. This can be partially attributed to the fact the StrongDAD does not provide a mechanism to handle a situation where multiple nodes request for the same address simultaneously. High rate of node admission is likely to lead to more than one node joining and requesting for the same address at the same time.

In all the three protocols evaluated, the node admission rate has an impact on the address conflicts. It is therefore imperative to adopt a first come first served mechanism in the allocation of IP addresses. Although this method results in high latency, the positive effect towards the configuration on unique addresses is very crucial. The adaptation mechanism employed in D-DAD did not address this issue.

6.4 Effect of node density

This experiment was performed to investigate the impact of interference on the performance of the D-DAD protocol. To investigate the impact of transmission interference on the proposed algorithm, node density was varied. Some studies on routing protocols show that with a very sparsely populated network the number of possible connections between any two nodes is low and hence the performance is poor. It is also noted that as the node density is increased the throughput of the network increases. However, beyond a certain level of node density, the performance starts to degrade (Varshney et al., 2016). This experiment investigated if the node density has an effect on the performance of address allocation protocols. The number of nodes was fixed at 70.

Address uniqueness and communication overhead for the three protocols were recorded and analysed.

Table 6.3 shows the other simulation parameters. As in experiments I and 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 was always constant for the duration of the simulation. Also, for the same reason, there were no node departures and mobility for the entire duration of the simulation.

Table 6. 3 : Simulation parameters for experiment III

Parameters	**Environment**
Number of nodes	70
Preconfigured nodes	1
Area	$(500m, 600,700, \ldots 1200m)^2$
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

i. Effect of node density on Address uniqueness

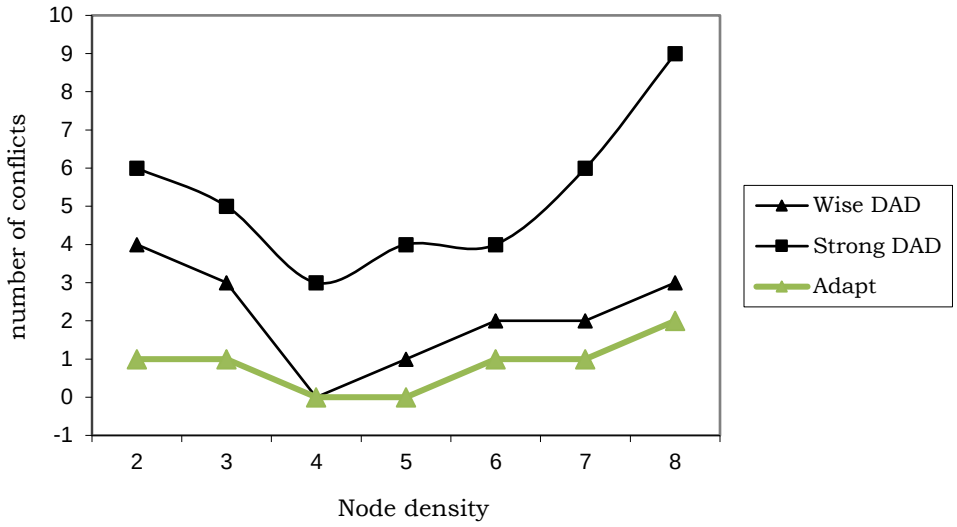

Figure 6. 7: Address conflicts against node density

Figure 6.7 shows the number of address conflicts against the node density. For the three protocols, with the node density around 4 or 5, the number of address conflicts were at their lowest. This can be attributed to the fact that the value of 4 has been found to be the optimal number of neighbours that achieve the best performance in the network. Node density affects interference, which in-turn affects message delivery (Mudali et al., 2007). Messages from neighbours are received free of errors provided that only one neighbour is transmitting (Borbash et al., 2007). D-DAD recorded the lowest number of address conflicts, due to its robustness. StrongDAD on the other hand had the largest number of address conflicts recorded.

The effect of node density has been investigated in literature – The results points to the inclusion of topology control algorithms within address allocation mechanisms. This has not been investigated but results calls for such inclusion. The fact that the CNN recorded in the investigation of the effect topology on routing protocol indicate that this is crucial and possible to implement. [this at the end of the sub section – check also the contributions]

ii. Node density on communication overhead

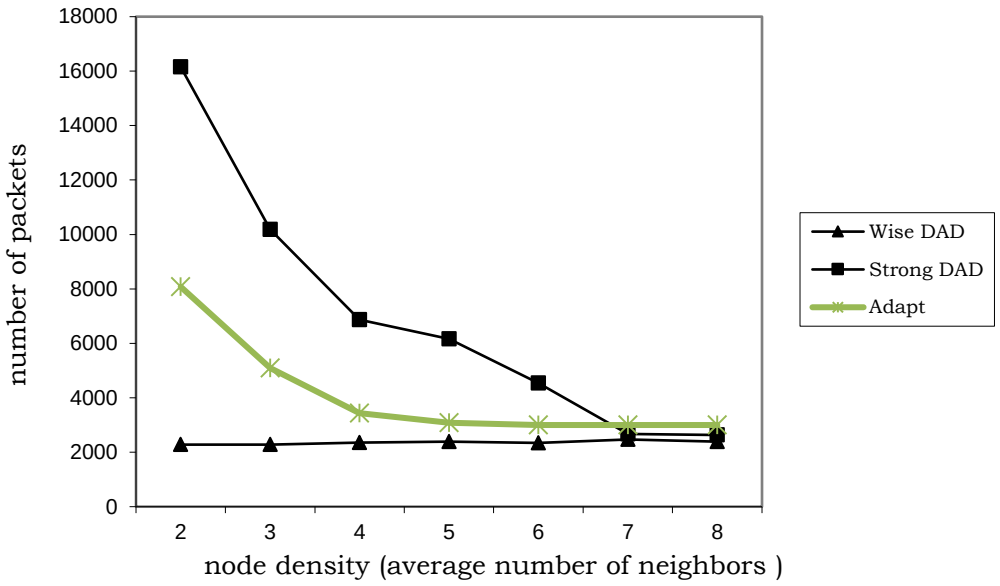

Figure 6. 8: Communication overhead against node density

The number of packets recorded in Wise-DAD was not affected by node density, but StrongDAD and D-DAD had high communication overhead when the node density was low. The communication overhead recorded in D-DAD simulation decreased and remained constant at the node density of 5 neighbours. The decrease in communication overhead in

StrongDAD and D-DAD was due to the fact that as node density increases, the number of links amongst the nodes increase as well. Increase in the number of links implies that fewer packets are re-broadcasted.

6.5 Effect of Network traffic on Address Auto-configuration

The purpose of this experiment was to investigate the effect of network size on the proposed address allocation protocol in the presence of network traffic. In each case, 20 nodes were introduced into an already existing network that had nodes already communicating. No address duplicates existed before the new nodes were introduced. IP address configuration delay, communication overhead and the number of address conflicts were recorded.

a. Effect of network traffic on latency

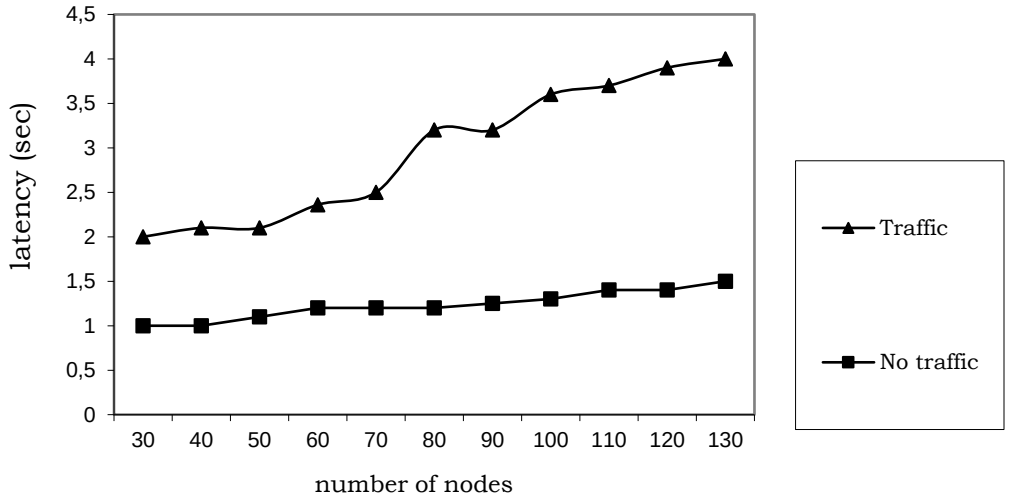

Figure 6. 9: latency against number of nodes

The results shown in Figure 6.9 illustrate the effect of network size on latency in the presence of network traffic. In both experiments, address allocation latency increased proportionally to network size. This is due to the fact that network traffic increased with network size, thereby affecting the delivery of address allocation packets. Increase in traffic caused an increase in latency of address allocation packets. When network traffic was introduced, the latency slightly increased for all network sizes. DAD timeout period was set at 1.4 seconds and each new node performed only one DAD trial. In the presence of network traffic D-DAD performed DAD more than once, resulting in latency of at least 2 seconds on each address allocation. Any value more than 3 seconds was caused by delay in delivering address allocation packets. Message delays were a result of the network size and the network traffic.

b. Effect of network traffic on address uniqueness

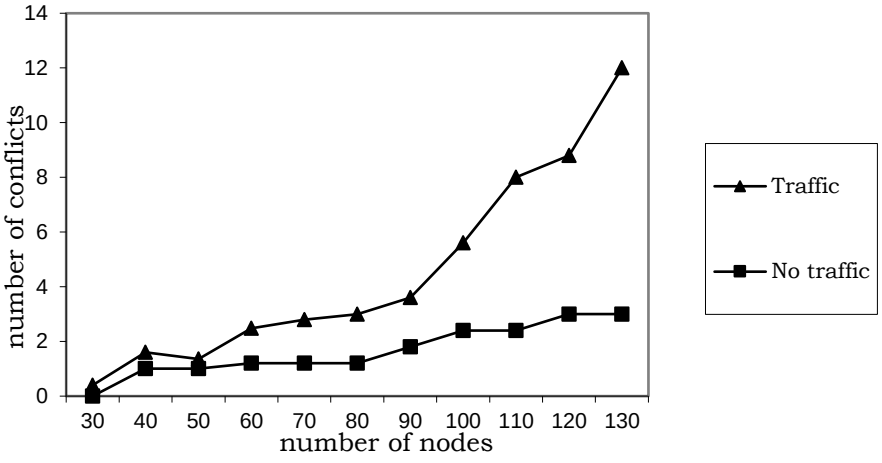

Figure 6. 10: Address duplicates against number of nodes

Figure 6.10 above shows the number of address duplicates against the number of nodes. When the network had no traffic, the number of address duplicates was low. As the number of nodes increase, the difference between the duplicates recorded in the two experiments increase. The number of duplicates recorded in the presence of network traffic increases at a faster rate than in the absence of network traffic. This can be attributed to the fact that network traffic may have negatively affected the delivery of address allocation packets before the expiry of DAD. If address allocation packets are not delivered, address duplicates are bound to occur since nodes will not be able to successfully defend their IP addresses. In D-DAD, once the DAD timeout period expires, the requested IP address is configured. D-DAD used a DAD timeout period of only 1 second. When there is network traffic, the DAD timeout might expire before an address allocation packet reporting an address conflict is received.

c. Effect of network traffic on communication overhead

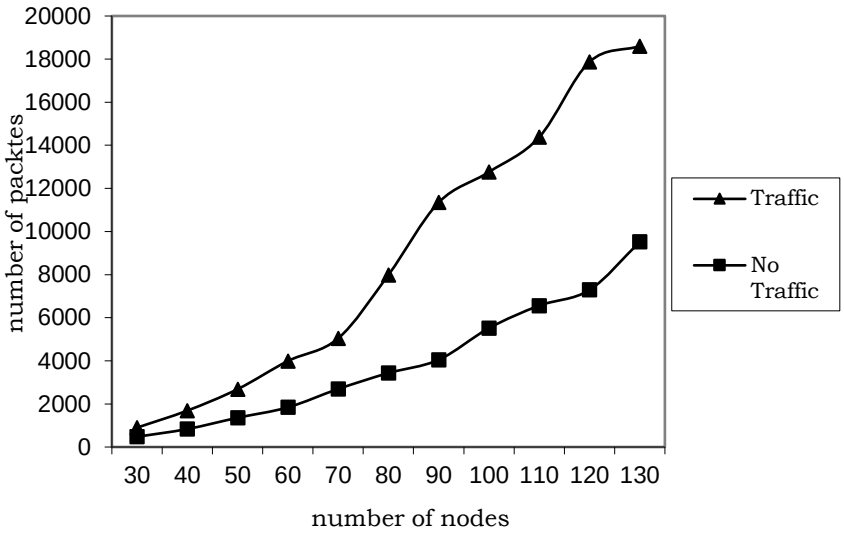

Figure 6. 11: communication overhead against number of nodes

The result shown in Figure 6.11 shows that the number of packets is in proportion to the number of nodes in both experiments. When traffic was introduced, the number of packets increased significantly. This is due to the fact that the protocol uses network- wide broadcast in the initial stages of address solicitation. This broadcast, however, is kept to a minimum due to the local address table used by the protocol. In the presence of network traffic, address allocation packets are higher, due to the increase in address duplicates, as shown in Figure 6.10. When address conflicts increase, the process of resolving address conflicts is activated, hence the increase in communication overhead.

6.6 Effect of Mobility on the D-DAD protocol

This section presents the experimental results obtained from the analysis of the effect of mobility on the D-DAD protocol. The simulation experiments analysed the performance of D-DAD under the Random Way Point Mobility model.

a. Effect of mobility on communication overhead

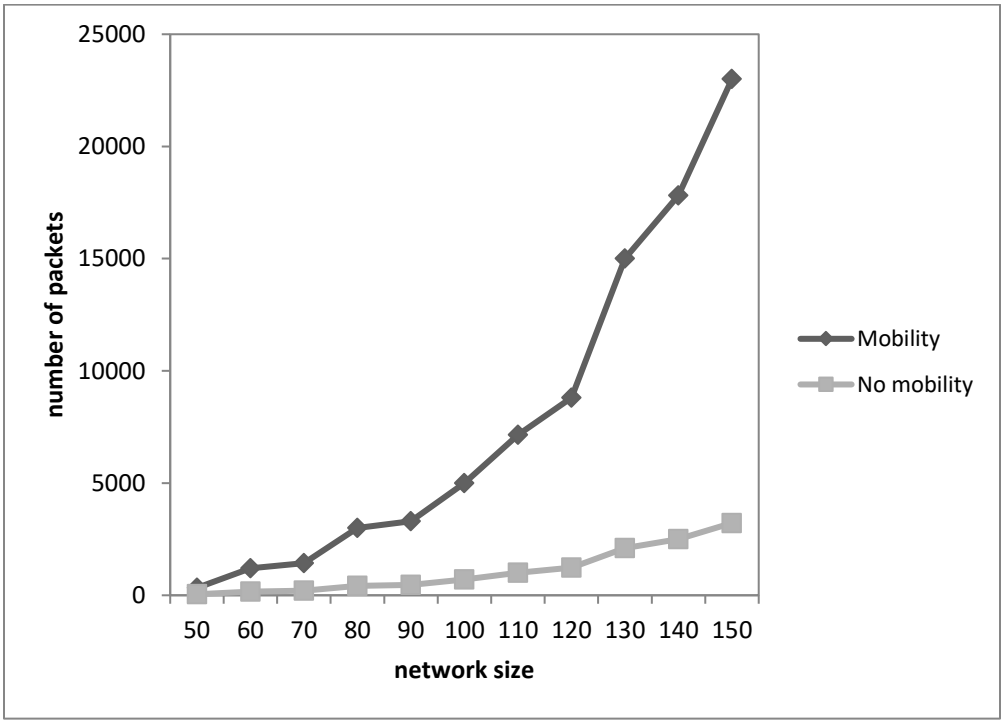

Figure 6. 12: number of packets against network size

Figure 6.12 above shows the impact of mobility on communication overhead during address auto- configuration. Communication overhead was found to be proportional to the network size. After mobility was introduced, a rapid increase in communication overhead

was observed. The increase in the communication overhead usually results in a high rate of packet loss, hence more address conflicts. The proposed protocol however, was more robust than StrongDAD, hence from Figure 6.12 below, the address conflicts recorded were not as high as those of StrongDAD in Figure 4.8, chapter 4.

b. Effect of mobility on latency

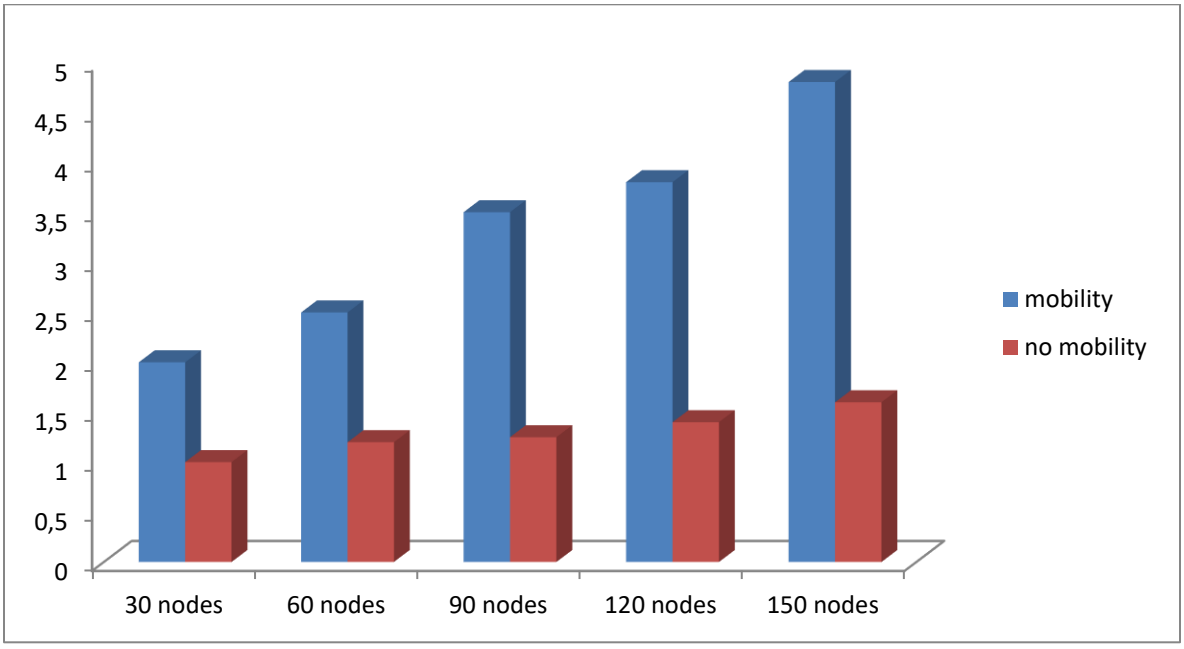

Figure 6. 13: Latency against network size

Figure 6.13 above presents the results of latency recorded when the D-DAD was subjected to mobility. For smaller network sizes, node mobility had little impact on latency but as the network size increased, the recorded latency increased exponentially. This is due to the fact that when D-DAD detects mobility it performs more network- wide broadcasts and increases the DAD timeout period from the minimal value of 1 second to the maximum value of 1.4 seconds. The combination of these two factors affects latency.

c. Effect of mobility on address uniqueness

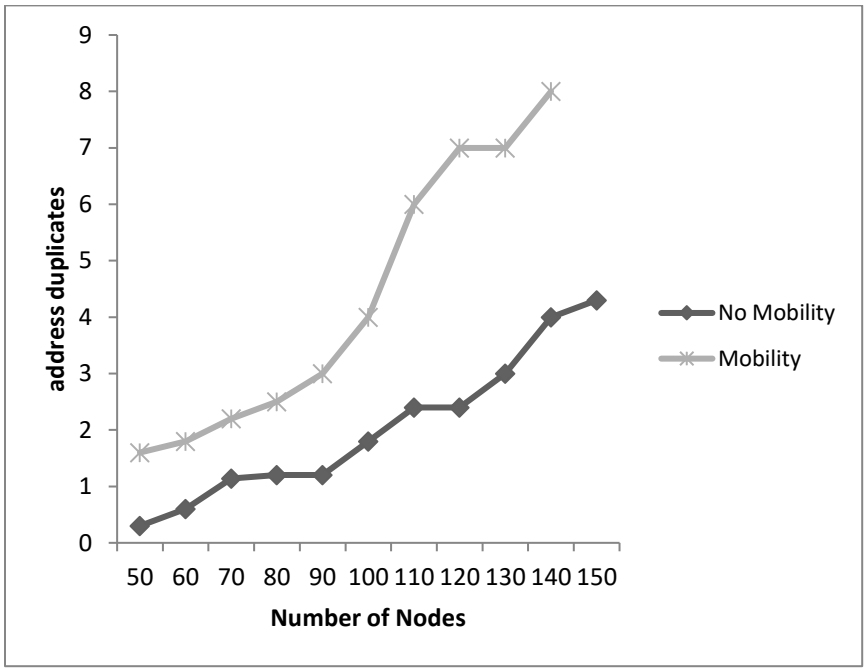

Figure 6. 14 address duplicate against network size

Figure 6.14 presents a comparison of the number of address conflicts for a network with and without mobility. To gain a comprehensive understanding, we varied the network size from 50 nodes to 150 nodes. In the presence of mobility, the address duplicates increased substantially. Address duplicates increased proportionally with network size. From the results it is clear that network size has a significant impact on the number of address duplicates, as they increase rapidly with network size. This is despite the fact that D-DAD protocol adopted a network-wide broadcast of address allocation messages in a bid to recover from possible message losses.

6.7 Effect of network partitioning on overhead

In this section, we investigated the amount of communication overhead generated during the detection of network partitioning. We did not perform simulations in which nodes transfer data from the application layer, because the main goal was assessing the traffic generated by the proposed solution independently from upper layers. When a network is partitioned into two, the partitioning should be detected so that addresses from the other segment of the network can be re-used. In the experiment, the number of control packets require d to detect and manage network merging were monitored. Figure 1 shows the amount of communication overhead generated during the partitioning of the network. Networks of nodes between 10 and 100 were used. The networks were divided into two equal partitions by slowly moving half of the nodes away from their original positions. The moving nodes were kept close to each other so as to avoid further partitioning. From the results obtained, the number of nodes in the network was directly proportional to the communication overhead. Periodic messages containing the list of **K** are broadcast, hence communication overhead is generated before the partitioning takes place. Soon after partitioning is detected the nodes constantly monitor their neighborhoods to check if the lost partition is back or not. This also generates more control packets. For a 100 node network, the communication overhead was as high as 4000 packets.

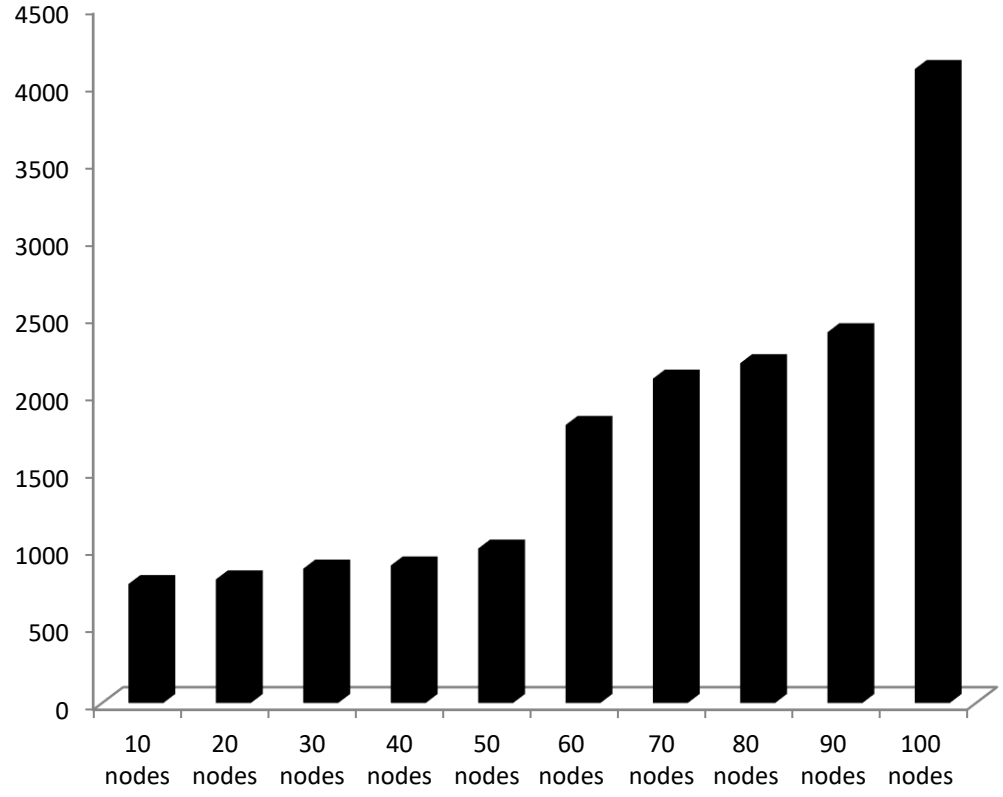

Figure 6. 15: Communication overhead during network partitioning

Figure 6.16 shows the amount of communication overhead generated during address resolution. The graph shows the amount of communication overhead versus the number of duplicates in the network. In this experiment, two networks of 100 nodes were merged. At the beginning of the experiment the networks were separated and configured separately. After 10 seconds, mobility was introduced, merging the two networks. On merging, the two networks had preset duplicate addresses. These were varied from 10 to 100 as shown in Figure 6.15.

A lot of communication overhead was generated when the number of duplicates was very high, due to the fact that a lot of nodes had to relinquish their IP addresses and acquire new

ones. The amount of traffic generated, however, is necessary for eliminating duplicate addresses. When the preconfigured duplicate addresses are fewer, the amount of communication overhead does not significantly affect bandwidth of the network.

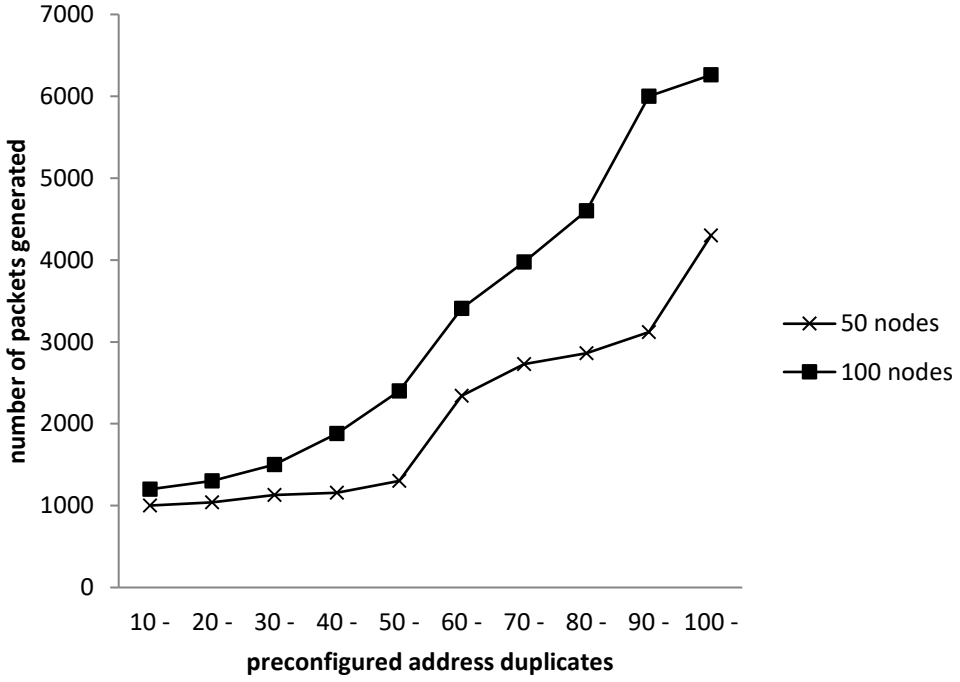

Figure 6. 16: Communication overhead during network merging

6.8 Resolution of Address Duplicates

The goal of this experiment was to determine the ability of the proposed protocol to resolve the issue of duplicate addresses after network merging.

In the experiment, two networks of 100 and 50 nodes were used. On merging, the networks had preset duplicate addresses. These were varied from 5 to 50, as shown in Figure 6.17. The network merging process was invoked and the address resolution procedure was allowed to take place. At the end of the experiment the number of address duplicates was

recorded. When the preconfigured duplicate addresses were set to 5 and 10, the merging mechanism managed to resolve all of them. As the number of preconfigured duplicate addresses was increased, the number of address conflicts also increased.

This is due to the fact that when duplicate addresses have been detected, all nodes with duplicate addresses start acquiring new addresses, thereby clogging the network with address configuration packets. As the amount of traffic increases so does the number of address duplicates.

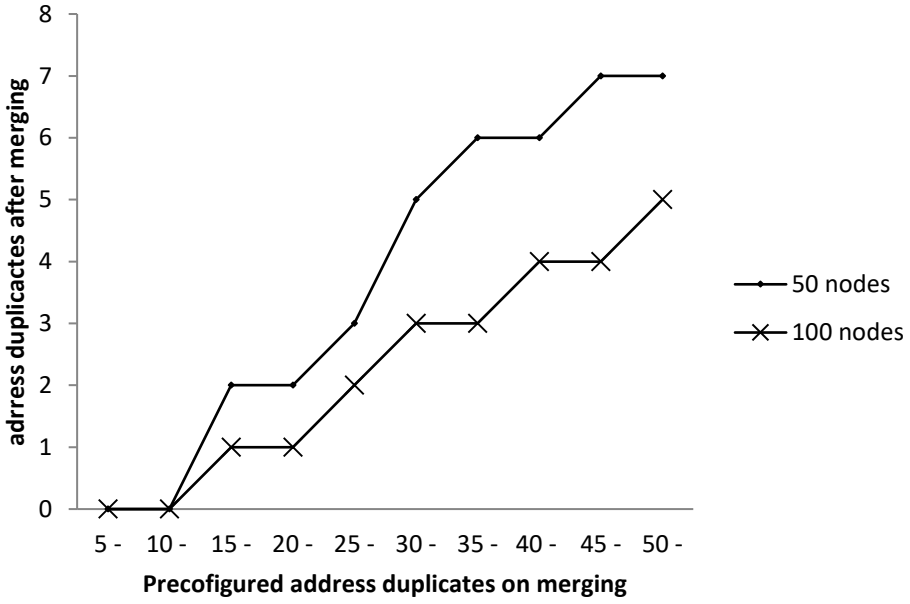

Figure 6. 17: Address duplicates after network merging

6.9 Latency of the Network merging process

This experiment was aimed at evaluating the latency of the network merging process. In the experiment, the number of nodes was varied from 10 to 100, as shown in Figure 6.18. In this experiment, two networks of 100 nodes were merged. At the beginning of the experiment, the networks were separated and configured separately. After 10 seconds, mobility was introduced, merging the two networks. On merging, the two networks had preset duplicate addresses. The number of duplicate addresses was set at 10% and 20% of the network size after merging. The network merging process was invoked and the address resolution procedure was allowed to take place. At the end of the experiment, average latency was recorded.

The latency of the address configuration did not change much as the number of nodes was increased. At 10% address conflicts, the latency slightly decreased as the number of nodes was increased. This can be attributed to the lower amount of communication overhead in relation to the network size. 20% address conflicts recorded slightly more latency that 10% address conflicts.

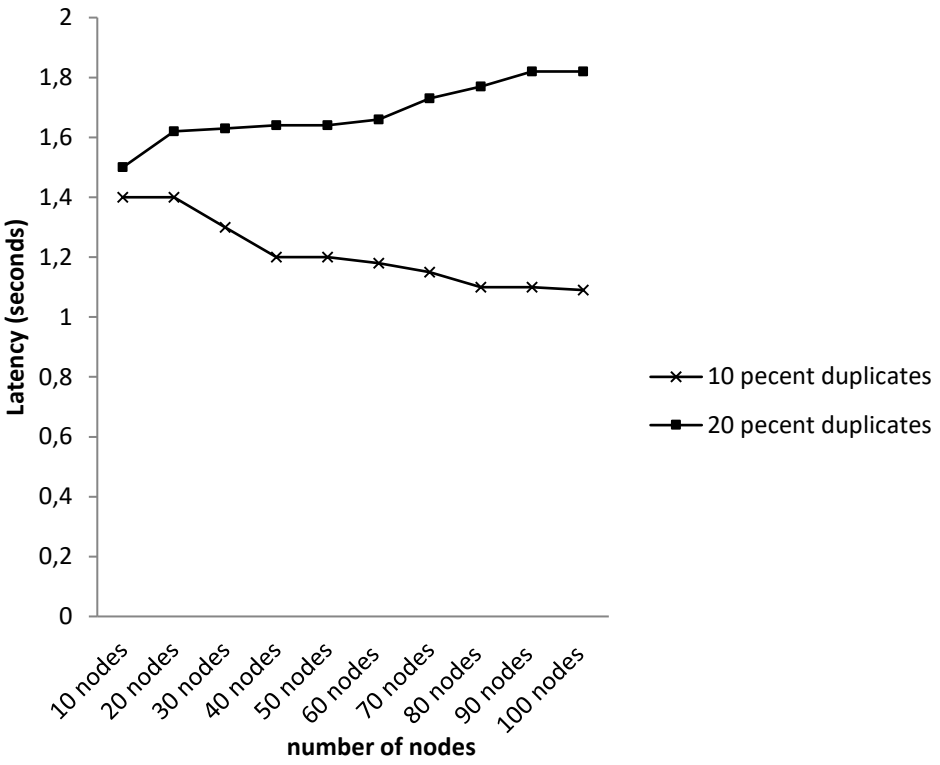

Figure 6. 18: Latency on address resolution

6.10 Chapter Summary

This chapter presented simulation results of the experiments conducted to test the proposed algorithms. To gain a comprehensive understanding of the performance of the proposed algorithm, a total of 8 experiments were conducted. The proposed algorithms were subjected to different network conditions and results were graphically presented and analysed. The design goal of our solution was to build a protocol that can react to network conditions rather than set static values for the DAD timeout period and the number of DAD trials. The proposed solution was inspired by swam intelligence hence has adaptation capabilities. As a result, static value of DAD timeout period does not give the best performance for address auto-configuration. This can further be supported by experiments reported in chapter 3. The use of adaptive values of DAD timeout period gave the network stability even when the network conditions were varied.

In the first experiment conducted in this chapter, we investigated the effect of network size on the proposed protocol in order to test its scalability. D-DAD recorded better latency and address conflicts than Wise-DAD and StrongDAD. However, the amount of communication overhead recorded was relatively high when compared to Wise-DAD. In the second experiment, we investigated the effect of node arrival rate on the D-DAD. The D-DAD performed better than Wise-DAD and StrongDAD on all the three metrics used for comparison. In the third experiment, node density was found to have an effect on address allocation. Node density of 4 was found to result in the best performance for the three protocols investigated.

The effect of node mobility and network traffic was also investigated. The results show that both mobility and network traffic have a negative effect on the performance of the D-DAD. However, the D-DAD protocol adjusted to node mobility and network traffic, hence the address conflicts observed were not as high as values observed in StrongDAD. From the result of this experiment we can conclude that there is a close relationship between node mobility and performance of address auto-configuration protocols. We also performed experiments to test the network merging and partitioning algorithms that are part of the DDAD auto-configuration protocol. The results obtained in the experiments clearly show that the adoption of swam intelligence as a solution approach brought stability. Changes such as network traffic, mobility and network sizes did not have a huge impact on the performance of the proposed protocol due to the adaptive nature of swam systems.

Chapter 7

Thesis Conclusion, Contributions and Future Work

7.1 Conclusion

This study represents a successful attempt to investigate automatic IP addressing in wireless ad hoc networks. The investigation first answered the question of how different network conditions affect the auto-configuration process. Experiments to investigate the effect of mobility, network traffic and DAD timeout period on addressing auto-configuration were conducted. The results of this investigation informed the design of a model inspired by swam intelligence. Consequently, a new protocol, D-DAD was proposed and evaluated in this thesis. Addressing the issues identified in Chapter 1, among other things, helped in building a robust IP addressing protocol and provide QoS guarantees in the network.

In order to tackle the identified research issues, the following research questions were formulated:

i. Why are existing paradigms not handling network dynamics well?
ii. Why are address auto-configuration algorithms not resilient enough to withstand different network conditions?
iii. What are the best configurations for DAD that can result in low address conflicts and low communication overhead?

iv. How can the wireless ad hoc networking environment inform the design of address auto-configuration protocols that can adapt to different network conditions?

An investigation that was reported in chapter 2 concluded that the existing paradigms are rigid and fail to adequately address the address allocation problem because of the unpredictable nature the wireless ad hoc networking environment. This conclusion led to the proposal of a new model inspired by swam intelligence presented in chapter 2. The model views the problem of IP addressing as that of achieving desired emergent behavior in the midst of conflicting objectives and criterion. As a result, the model presented advocates for, among other things, continuous monitoring of network conditions such as mobility, network traffic and node density. The solution prosed later considers the multiple objectives that an address allocation scheme should consider when configuring addresses. Therefore, chapter 2 answered the first research question.

To answer the second research question, the following investigations were conducted:

i. Determining the effect of network merging and partitioning on address auto-configuration.
ii. Determining the effect of network traffic and mobility on address auto-configuration.

The results obtained from the investigations of the second research question were used to develop the address auto-configuration protocol proposed in this thesis. We investigated

the factors that affect the DAD protocol before we applied the results of the investigation to construct the building blocks for the proposed solution. Simulations to determine the optimal value of the DAD timeout period were also conducted. This effectively answered the third research question.

To address the fourth question, we investigated how the results from the first three questions could be used in the design of Address Auto-configuration protocols. This led to the design of the D-DAD protocol. We compared the D-DAD with the Wise-DAD and StrongDAD protocols. We performed five experiments to investigate the effect of network size, node density, node arrival rate, mobility, and network traffic on communication overhead, address uniqueness and latency. The results showed that D-DAD outperformed StrongDAD in all the metrics used for comparison. However, in some instances, D-DAD recorded more communication overhead in comparison to Wise-DAD but had better latency and fewer address conflicts.

In the first experiment, D-DAD showed better scalability since it performed better than both Wise-DAD and StrongDAD when network size was increased. However, communication overhead recorded in D-DAD was slightly higher than Wise-DAD but the number of IP address duplicates recorded was low.

From the results of the second experiment, it was observed that D-DAD did not show any change in performance as node arrival rate was varied. On the other hand, the number of address duplicates in StrongDAD decreased as the node arrival rate was increased. Interference significantly affected communication overhead recorded in StrongDAD. 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. In the last experiment we observed that node density has a negative effect on address allocation.

There are various conclusions that can be drawn from this work:

(a) First, a static value of the DAD timeout period does not give the best performance for address auto-configuration. (This investigation is reported in Chapter 3). The changing network conditions require that the protocol adjust the number of DAD trials and the DAD timeout period. This conclusion motivated the design of a mechanism where the DAD timeout period is determined at runtime. This allowed the new protocol to adapt to different network conditions.

(b) Investigations reported in Chapter 3 concluded that node mobility, no matter how low, has a negative effect on address allocation protocols. Mechanisms that handle mobility on routing protocols may be explored as possible solutions to node mobility in address allocation protocols. The investigation carried out in this thesis found a close relationship between node mobility and performance of address auto-configuration protocols.

(c) From the investigation reported in Chapter 4 this thesis can conclude that the use of an initiator is not suitable in a network that has high mobility. If an initiator or the new node moves, the address allocation process is severely affected. Mechanisms of mobility detection need to be incorporated into address allocation protocols. The idea of initiator has been adopted by many protocols in the literature despite this shortcoming.

(d) Node density has an effect on the performance of address allocation protocols. A lot of work on the effect of node density on routing protocols exists in the literature. This work has concluded that node density also affects address allocation. Investigations reported in Chapter 6 concluded that node density of 5 neighbours resulted in the optimal performance of address allocation protocols.

7.2 Thesis Contributions

This thesis has investigated the problem of address allocation in wireless ad hoc networks. Below, an explanation of the contributions made in this thesis is given.

(a) Although a lot has been done in the area of address auto-configuration for wireless ad hoc networks, no work to date has, to the best of our knowledge, considered the issue of adapting to network conditions, which was accomplished in this work. Most importantly is the idea of adapting from the angle inspired by swam systems. Current proposals set constant parameter values for DAD timeout period and number of DAD trials. This thesis, on the other hand, has proposed a paradigm shift to the problem of IP address auto-configuration. The newly proposed paradigm in this work advocates for adaptation whereby parameters such as DAD timeout period are determined at runtime. The thesis has further argued that address allocation protocols should have monitoring mechanisms to monitor for conditions that affect the functioning of the protocol. In addition, the new paradigm proposes the use of adaptation mechanisms to respond to any situation that may affect the functioning of the protocol. Traditionally, address allocation protocols are categorised as stateless, stateful and hybrid. Stateless auto-configuration uses the trial and error method to obtain a free address. Stateful protocols use address allocation tables, meaning that free addresses are known in advance. Hybrid protocols combine the characteristics of both. The approach proposed in this work attempts to build a system that converges towards the global desired goal.

(b) This thesis has advanced the design of DAD by establishing the optimal values for the DAD timeout period. The study is also, to the best of our knowledge, the first to determine the minimal value for DAD timeout period to be 1 second whilst the maximal value was determined to be 1.4 seconds. Current DAD-based protocols use 1.8 seconds, which is calculated from the estimate of 12 as the maximum hop count, thus the timeout must be at least 1.8 seconds (Kim et al., 2007).

(c) It is already known that network conditions affect network performance in wireless ad hoc networks (Alvarez et al., 2016; Nayak & Vathasavai, 2016). What is not clear is the effect of network conditions on the address auto-configuration process. Conditions such as network topology and mobility, have been found to have an effect on the performance of wireless ad hoc networks (Rao & Singh, 2015). While much effort has been put into the development of new IP address auto-configuration protocols for wireless ad hoc networks very little has been done in testing how different network conditions affect the performance of these protocols. The dynamic change of network conditions in wireless ad hoc networks means that more sophisticated protocols are desirable. This thesis has extended the solution space of address allocation protocols by investigating the effect of network traffic and mobility of address allocation protocols. The results obtained in the investigations compel researchers to look at address allocation solutions differently. It is evident from the results obtained that more robust protocols that take mobility and network

into account are needed. Current solutions do not consider this important aspect in their design.

(d) If a network partitions into two, the nodes need to detect this occurrence and either generate a new network ID or remain with the same network ID. How this process is handled is still an open research area within auto-configuration of IP addresses (Lee et al., 2015). In other instances, network partitioning may be temporary, due to poor links and dynamic network membership. The auto-configuration protocol must be able to distinguish between temporary and permanent network partitioning. Contrary to proposals in the literature, the network partitioning proposed in this thesis can distinguish between temporary and permanent partitions. Being able to distinguish between temporary and permanent partitions removes the burden of unnecessary address and network ID changes which can cause a lot of communication overhead.

(e) Network merging solutions have been investigated extensively and new solutions proposed but, to date, apparently no work has considered the merging of two networks that were previously combined. As an improvement to most solutions proposed in the literature, the solution proposed in this thesis handles the merging of networks that were previously part of the same network without changes in IP addresses.

(f) Although the effect of node density on wireless ad hoc network performance has been investigated extensively with respect to routing protocols (Younis et al., 2014; Zhao, 2014), the same cannot be said about wireless ad hoc networking with respect to address auto-configuration protocols. Node density has been an important aspect to be considered when planning the deployment of ad hoc networks (Barrachina et al., 2015).The results in this thesis found a relationship between node density and the performance of the address auto-configuration protocol. A node density of 5 was found to produce the optimal performance. These findings are important as they can be used when planning node placement for network deployment where node placement is of paramount importance.

7.3 Limitations and Future Work

This section presents an evaluation of the work presented in this thesis. Shortcomings and possible directions for future work are explained. Some of the concerns and limitations highlighted in this section may not be directly related to the problem statement or research questions, but their importance to IP address auto-configuration cannot be ignored:

(a) The proposed solution, D-DAD, enhances the procedure of IP address configuration but problems such as security still need to be completely worked out. In this work, we assumed that the nodes joining the network are not malicious, hence the issue of security was not discussed.

(b) The computational complexity of the algorithms designed was not evaluated. However, the design was kept as simple as possible. In any resource-constrained environment, the issue of resource management is of paramount importance. Thus, the memory and processing requirements of the proposed solution are other important issues needing further investigation. It is envisaged that if such investigations are done auto-configuration protocols could be lightweight enough to be deployed in handheld and other capacity-constrained devices.

(c) In Chapter 5, the network partitioning solution proposed requires the selection of a set of nodes, **K**. Network partitioning is detected once a certain portion of **K** is missing. This work did not determine the exact value of the subset **K** that should be missing for network partitioning to be detected.

(d) The auto-configuration protocol proposed in this work is designed to adapt to network traffic. However, the detection of traffic volume and type was not considered. Nodes were assumed to run some algorithm that allows them to intelligently monitor the type and volume of network traffic present in the network. For illustration purposes, the proposed protocol analysed the rate of traffic flow on a given node. In real-life scenarios, this assumption may not be acceptable.

(e) The experiments conducted in this work were done in the NS2 simulator. To obtain more realistic results, it is desirable that experiments be conducted in real-life testbed scenarios. However, to obtain any meaningful results for scenarios such as network merging requires a large number of physical devices. Acquiring large numbers of such devices was not feasible in this work since it would have required extra financial resources and time not budgeted for.

(f) Another important part of this work that would have been challenging to investigate on a testbed is node mobility. It is against this background that we chose, in this work, to limit the proof of concept to simulation.

BIBLIOGRAPHY

Abid, S. A., Othman, M., Shah, N., Sabir, O., Khan, A. ur R., Ali, M., … Ullah, S. (2015). Merging of DHT-based logical networks in MANETs. *Transactions on Emerging Telecommunications Technologies*, *26*(12), 1347–1367. https://doi.org/10.1002/ett.2969

Al-Maashri, A., & Ould-Khaoua, M. (2006). Performance Analysis of MANET Routing Protocols in the Presence of Self-Similar Traffic. In *Proceedings. 2006 31st IEEE Conference on Local Computer Networks* (pp. 801–807). IEEE. https://doi.org/10.1109/LCN.2006.322040

Alvarez, C. F., Palafox, L. E., Aguilar, L., Sanchez, M. A., Martinez, L. G., & Marenduzzo, D. (2016). Using Link Disconnection Entropy Disorder to Detect Fast Moving Nodes in MANETs. *International Journal of Simulation--Systems, Science & Technology*, *17*(34), 14–31. https://doi.org/10.1371/journal.pone.0155820

Ancillotti, E., Raffaele, B., Marco, C., & Antonio, P. (2009). Dynamic address autoconfiguration in hybrid ad hoc networks. *Pervasive and Mobile Computing*, *5*(4), 300–317. https://doi.org/10.1016/J.PMCJ.2008.09.008

Barrachina, S., Adame, T., Bel, A., & Bellalta, B. (2015). GOAT: A Tool for Planning Wireless Sensor Networks. In *In International Workshop on Multiple Access Communications, Springer International Publishing* (pp. 147–158). Springer, Cham. https://doi.org/10.1007/978-3-319-23440-3_12

Bernardos, C., Calderón, M., & Moustafa, H. (2005). Survey of IP address autoconfiguration mechanisms for MANETs. *IETF, Draft-Bernardosmanetautoconf-Survey-05. Txt (Work-in-Progress)*. Retrieved from https://www.ietf.org/proceedings/69/slides/autoconf-13.pdf

Borbash, S., Ephremides, A., & McGlynn, M. (2007). An asynchronous neighbor discovery algorithm for wireless sensor networks. *Ad Hoc Networks*, *5*(7), 998–1016. Retrieved from http://www.sciencedirect.com/science/article/pii/S1570870506000278

Chin, W., Fan, Z., & Haines, R. (2014). Emerging technologies and research challenges for 5G wireless networks. *IEEE Wireless Communications*, *21*(2), 106–112. Retrieved from http://ieeexplore.ieee.org/abstract/document/6812298/

Dart, E., Beebee, W., George, W., Asati, R., Pignataro, C., & Singh, H. (2015). Enhanced Duplicate Address Detection. *Internet Engineering Task Force (IETF) Request for Comments: 7527*. Retrieved from https://tools.ietf.org/html/draft-ietf-6man-enhanced-dad-10

Divecha, B., Abraham, A., & Grosan, C. (2007). Impact of Node Mobility on MANET Routing Protocols. *Journal of Digital*, *5*(1). Retrieved from http://search.ebscohost.com/login.aspx?direct=true&profile=ehost&scope=site&authtype=crawler&jrnl=09727272&AN=24821549&h=%2FxSGoEzSZxjSPtDnourXhIr5JgkFIN5RXk9%2B%2FuKvsl8FjPtYS86AzVdTRvZsm79VWOe3qCNUK%2FwBFmxLzhAnfg%3D%3D&crl=c

Fan, Z., & Subramani, S. (2005). An address autoconfiguration protocol for IPv6 hosts in a mobile ad hoc network. *Computer Communications*, *28*(4), 339–350. https://doi.org/10.1016/j.comcom.2004.09.001

Fazio, M., Villari, M., & Puliafito, A. (2006). AIPAC: Automatic IP address configuration in mobile ad hoc networks. *Computer Communications*, *29*(8), 1189–1200. https://doi.org/10.1016/j.comcom.2005.07.006

Fernandes, N., & Moreira, M. (2013). An efficient and robust addressing protocol for node autoconfiguration in ad hoc networks. *IEEE/ACM Transactions on*. Retrieved from http://dl.acm.org/citation.cfm?id=2525555

Grajzer, M., & Głąbowski, M. (2016). Neighbor Discovery++: A Low-Overhead Address Auto-configuration to Enable Robust Internet of Things Architectures (pp. 87–97). Springer, Cham. https://doi.org/10.1007/978-3-319-28561-0_7

Güne, M., & Reibel, J. (2002). An IP Address Configuration Algorithm for Zeroconf. Mobile Multi-hop Ad-Hoc Networks *. *International Workshop on Broadband Wireless Ad-Hoc Networks and Services*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.68.6376&rep=rep1&type=pdf

Harish, K., Singla, R. K., & Malhotra, S. (2008). Issues & Trends in AutoConfiguration of IP Address in MANET. *Int. J. of Computers, Communications & Control*, *III*, 1841–9836. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.567.1339&rep=rep1&type=pdf

Hossain, E., Rasti, M., Tabassum, H., & Abdelnasser, A. (2014). Evolution toward 5G

multi-tier cellular wireless networks: An interference management perspective. *IEEE Wireless Communications*, *21*(3), 118–127. https://doi.org/10.1109/MWC.2014.6845056

Indrasinghe, S., Indrasinghe, S., Pereira, R., & Mokhtar, H. (2006). Hosts Address Auto Configuration for Mobile Ad Hoc Networks. *4th International Conference on Performance Modeling and Evaluation of Heterogeneous Networks.* Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.95.8128

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*, *30*(8), 1913–1925. https://doi.org/10.1016/j.comcom.2007.03.002

Kim, S.-M., Choi, H.-S., & Rhee, W.-S. (2015). IoT home gateway for auto-configuration and management of MQTT devices. In *2015 IEEE Conference on Wireless Sensors (ICWiSe)* (pp. 12–17). IEEE. https://doi.org/10.1109/ICWISE.2015.7380346

Kumar, S., Agrawal, G. S., & Sharma, S. K. (2015). *Impact of Node Mobility on MANETs Routing Protocols under Random Waypoint, Group and File Mobility Models*. *INROADS- An International Journal of Jaipur National University* (Vol. 5). Sekiyu gakkai. Retrieved from http://www.indianjournals.com/ijor.aspx?target=ijor:inroads&volume=5&issue=1s&article=044

Lee, S., Younis, M., & Lee, M. (2015). Connectivity restoration in a partitioned wireless sensor network with assured fault tolerance. *Ad Hoc Networks*. Retrieved from http://www.sciencedirect.com/science/article/pii/S1570870514001437

Levin, L., Efrat, A., & Segal, M. (2014). Collecting data in ad-hoc networks with reduced uncertainty. *Ad Hoc Networks*, *17*, 71–81. Retrieved from http://www.sciencedirect.com/science/article/pii/S157087051400016X

Loo, J., Jaim, L. M., & Jesus, H. (2016). *Mobile Ad Hoc Networks: Current Status and Future Trends - Google Books*. CRC Press, Taylor & Francis Group, Boca Raton London New York. Retrieved from https://books.google.com.ng/books?hl=en&lr=&id=k-zRBQAAQBAJ&oi=fnd&pg=PP1&dq=Mobile+ad+hoc+networks:+current+status+and+future+trends&ots=ajjOls4jnw&sig=dJNXc3DOkKdWfxXfxiRRlDYAS_Q&redir_esc=y#v=onepage&q=Mobile ad hoc networks%3A current status

Mennicken, S., Vermeulen, J., & Huang, E. M. (2014). From today's augmented houses to tomorrow's smart homes. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing - UbiComp '14 Adjunct* (pp. 105–115). New York, New York, USA: ACM Press. https://doi.org/10.1145/2632048.2636076

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* (pp. 1–6). IEEE. https://doi.org/10.1109/ATNAC.2012.6398066

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*.

Nayak, P., & Vathasavai, B. (2016). Impact of Random Mobility Models for Reactive Routing Protocols over MANET. *International Journal of Simulation--Systems, Science & Technology*, *17*(34), 14–31. https://doi.org/10.5013/IJSSST.a.17.34.13

Nesargi, S., & Prakash, R. (2012). MANETconf: configuration of hosts in a mobile ad hoc network. In *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies* (Vol. 2, pp. 1059–1068). IEEE. https://doi.org/10.1109/INFCOM.2002.1019354

Nitnaware, D. (2016). Investigating the Performance of Energy Efficient Routing Protocol for MANET Under Pareto Traffic (pp. 145–152). Springer, Cham. https://doi.org/10.1007/978-3-319-30927-9_15

Pan, W., Reeves, D. S., & Ning, P. (2005). Secure address auto-configuration for mobile ad hoc networks. In *The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services* (pp. 519–521). IEEE. https://doi.org/10.1109/MOBIQUITOUS.2005.52

Pathan, A.-S. K. (Ed.). (2016). *Security of Self-Organizing Networks: MANET, WSN, WMN, VANET - Google Books*. CRC Press, Taylor & Francis Group, Boca Raton London New York. Retrieved from https://books.google.com.ng/books?hl=en&lr=&id=ZtBnZoijaDcC&oi=fnd&pg=PP1&dq=security+of+self-organizing+networks+manet+wsn+wmn+vanet&ots=cMR8wip_eb&sig=JsfKHiimPuKTPTs9IpF61FO-EfI&redir_esc=y#v=onepage&q=security of self-organizing

networks manet

Peffers, K., Tuunanen, T., Rothenberger, M. a., & Chatterjee, S. (2008). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, *24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Perkins, C., Malinen, T., Wakikawa, R., Belding-Royer, E., & Sun, Y. (2001). IP Address Autoconfiguration for Ad Hoc Networks. *IETF Internet Draft*. Retrieved from https://tools.ietf.org/html/draft-perkins-manet-autoconf-00

Pondwal, V., & Saini, H. (2016). 7 A Comprehensive Survey on Routing Schemes for High Speed Networks A Comprehensive Survey on Routing Schemes for High Speed Networks, *8*(4), 7–17. Retrieved from https://search.proquest.com/openview/5477994cf64f5968ac628ffc5d838077/1?pq-origsite=gscholar&cbl=2030006

Praptodiyono, S., Murugesan, R. K., Hasbullah, I. H., Wey, C. Y., Kadhum, M. M., & Osman, A. (2015). Security mechanism for IPv6 stateless address autoconfiguration. In *2015 International Conference on Automation, Cognitive Science, Optics, Micro Electro-Mechanical System, and Information Technology (ICACOMIT)* (pp. 31–36). IEEE. https://doi.org/10.1109/ICACOMIT.2015.7440150

Ramakrishnaiah, N., & Reddy, P. C. (2016). Tree based variable length address autoconfiguration protocol for mobile ad hoc networks. In *2016 2nd International Conference on Advances in Computing, Communication, & Automation (ICACCA) (Fall)* (pp. 1–6). IEEE. https://doi.org/10.1109/ICACCAF.2016.7748983

Rao, M., & Singh, N. (2015). Performance Evaluation of AODV nth BR Routing Protocol under Varying Node Density and Node Mobility for MANETs. *Indian Journal of Science and Technology*, *8*(17). https://doi.org/10.17485/ijst/2015/v8i17/70445

Rehman, S., & Manickam, S. (2015). Significance of Duplicate Address Detection Mechanism in Ipv6 and its Security Issues: A Survey. *Indian Journal of Science and Technology*, *8*(30). https://doi.org/10.17485/ijst/2015/v8i30/85940

Schoeneich, R. O., & Sutkowski, P. (2016). Performance of IP address auto-configuration protocols in Delay and Disruptive Tolerant Networks. *INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS*, *62*(2), 173–178. https://doi.org/10.1515/eletel-2016-0024

Sharma, P. (2013). Evolution of Mobile Wireless Communication Networks-1G to 5G as

well as Future Prospective of Next Generation Communication Network. *International Journal of Computer Science and Mobile Computing (IJCSMC)*, *2*(8), 47–53. https://doi.org/10.1046/j.1523-1739.1997.97069.x

Sibeko, N., Mudali, P., Oki, O., & Alaba, A. (2015). Performance evaluation of routing protocols in uniform and normal node distributions using inter-mesh wireless networks. In *2015 World Symposium on Computer Networks and Information Security (WSCNIS)* (pp. 1–6). IEEE. https://doi.org/10.1109/WSCNIS.2015.7368292

Suganthi, D., & Ravimaran, S. (2014). Collision Free Address Assignment for Nodes in Ad Hoc Networks Using FAP. *International Journal of Advanced Research in Computer Science and Electronics Engineering (IJARCSEE)*, *3*(6), pp:327-332. Retrieved from http://ijarcsee.org/index.php/IJARCSEE/article/view/489

Tan, D., & Kim, D. (2014). Dynamic traffic-aware routing algorithm for multi-sink wireless sensor networks. *Wireless Networks*. Retrieved from http://link.springer.com/article/10.1007/s11276-013-0672-z

Thriveni, H. B., Kumar, G. M., & Sharma, R. (2013). Performance Evaluation of Routing Protocols in Mobile Ad-Hoc Networks with Varying Node Density and Node Mobility. In *2013 International Conference on Communication Systems and Network Technologies* (pp. 252–256). IEEE. https://doi.org/10.1109/CSNT.2013.60

Vaidya, N. H. (2002). Weak duplicate address detection in mobile ad hoc networks. In *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing - MobiHoc '02* (p. 206). New York, New York, USA: ACM Press. https://doi.org/10.1145/513800.513826

Varshney, P., Agrawal, G. S., & Sharma, S. K. (2016). *Relative Performance Analysis of Proactive Routing Protocols in Wireless Ad hoc Networks using Varying Node Density*. *Invertis Journal of Science & Technology* (Vol. 9). OCTA études. Retrieved from http://www.indianjournals.com/ijor.aspx

Wang, J., Dong, W., Cao, Z., & Liu, Y. (2015). On the Delay Performance in a Large-Scale Wireless Sensor Network: Measurement, Analysis, and Implications. *IEEE/ACM Transactions on Networking*, *23*(1), 186–197. https://doi.org/10.1109/TNET.2013.2296331

Wang, X., Yang, Y., Yao, Y., & Cheng, H. (2014). An address configuration protocol for 6LoWPAN wireless sensor networks based on PDAD. *Computer Standards & Interfaces*, *36*(6), 918–927. https://doi.org/10.1016/j.csi.2014.02.006

Weniger, K. (2005). PACMAN: passive autoconfiguration for mobile ad hoc networks. *IEEE Journal on Selected Areas in Communications*, *23*(3), 507–519. https://doi.org/10.1109/JSAC.2004.842539

Weniger, K., & Zitterbart, M. (2004). Mobile ad hoc networks-current approaches and future directions. *IEEE Network*. Retrieved from http://ieeexplore.ieee.org/abstract/document/1316754/

Xu, L. Da, He, W., & Li, S. (2014). Internet of Things in Industries: A Survey. *IEEE Transactions on Industrial Informatics*, *10*(4), 2233–2243. https://doi.org/10.1109/TII.2014.2300753

Younis, M., Izzet, F. S., Kemal, A., Sookyoung, L., & Fatih, S. (2014). Topology management techniques for tolerating node failures in wireless sensor networks: A survey. *Computer Networks*, *58*, 254–283. https://doi.org/10.1016/J.COMNET.2013.08.021

Zakaria, E. E., Hamza, H. S., & Saroit, I. A. (2015). An Integrated Security Framework for Access Control and Address Auto-Configuration for MANETs. In *2015 8th IFIP Wireless and Mobile Networking Conference (WMNC)* (pp. 253–260). IEEE. https://doi.org/10.1109/WMNC.2015.18

Zhao, J. (2014). Minimum node degree and k-connectivity in wireless networks with unreliable links. In *2014 IEEE International Symposium on Information Theory* (pp. 246–250). IEEE. https://doi.org/10.1109/ISIT.2014.6874832

Zhou, H., Ni, L. M., & Mutka, M. W. (2013). Prophet address allocation for large scale MANETs. In *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)* (Vol. 2, pp. 1304–1311). IEEE. https://doi.org/10.1109/INFCOM.2003.1208966

Nag, D., Majumder, D., Raquib, C.M., Pramanik, S., Basu, A., Rana, T.K. and Rana, B., 2017, August. Green energy powered smart healthy home. In Industrial Automation and Electromechanical Engineering Conference (IEMECON), 2017 8th Annual (pp. 47-51). IEEE.

Yin, R.K., 2017. Case study research and applications: Design and methods. Sage publications.

Rana, M.K., Sardar, B., Mandal, S. and Saha, D., 2017. Implementation and performance evaluation of a mobile IPv6 (MIPv6) simulation model for ns-3. Simulation Modelling

Practice and Theory, 72, pp.1-22.

Ahmadvand, M. and Tamalloki, H., 2017. Using VIKOR method to prioritise sharia-compliant equivalents for short selling (based on evidence of Iran's stock market). Afro-Asian Journal of Finance and Accounting, 7(3), pp.281-303.

Jha, M., Seshadhri, C., & Pinar, A. (2015). A space-efficient streaming algorithm for estimating transitivity and triangle counts using the birthday paradox. ACM Transactions on Knowledge Discovery from Data (TKDD), 9(3), 15.

del Pilar Rios, A., Agbossou, K., & Cardenas, A. (2017, March). Taylor series approximation of ZIP model for on-line estimation of residential loads' parameters. In Industrial Technology (ICIT), 2017 IEEE International Conference on (pp. 632-637). IEEE.

Christenson, D.A. and Venuto, J., International Business Machines Corp, 2017. Duplicate IP address detection by a DHCP relay agent. U.S. Patent 9,774,487.

Rehman, S.U. and Manickam, S., 2016. Novel mechanism to prevent denial of service (DoS) attacks in IPv6 duplicate address detection process. Int. J. Secur. Appl., 10(4), pp.143-154.

Appendices

NS2 CODE FOR THE D-DAD protocol

Header File

```
#ifndef _DynamicDAD_H_
#define _DynamicDAD_H_

#include <ip.h>
#include <packet.h>
#include <random.h>
#include <timer-handler.h>
#include <agent.h>
#include <config.h>
#include <scheduler.h>

#define MAX_RETRY 3
#define RETRY_TIMEOUT 30 // 30 ms
#define DynamicDAD_PORT 224

#define CURRENT_TIME (Scheduler::instance()).clock()

// DynamicDAD address allocation agent
class DynamicDAD;

// A timer used by initiator to send reply

class DynamicDADTimer : public TimerHandler
{
	public:
		DynamicDADTimer(DynamicDAD* a) : TimerHandler(), agent(a) { }
		inline virtual void expire(Event*);
	private:
		DynamicDAD* agent;
};


class PartitionTimer : public TimerHandler
{
	public:
		DynamicDADTimer(DynamicDAD* a) : TimerHandler(), agent(a) { }
		inline virtual void expire(Event*);
	private:
		DynamicDAD* agent;
};

//  timer used by new node
class DynamicDAD_MembershipReqTimer : public TimerHandler
{
	public:
		DynamicDADAckTimer(DynamicDAD* a) : TimerHandler(), agent(a) { }
		inline virtual void expire(Event*);
	private:
		DynamicDAD* agent;
};
```

```
class DynamicDAD : public Agent
{
	friend class BroadcastTimer;
	public:
		DynamicDAD(void);
		void recv(Packet* p, Handler*);
		void SendAddressRequest(void);
		void SendAddressReply(nsaddr_t dest);
		void sendAddressConflict(nsaddr_t dest);
		void SendMembershipRequest(void);
		void sendCONFIRMATION(nsaddr_t dest);
		void SendInitiatorSelection(nsaddr_t dest);
		void SendNetworkDeparture(void);
		void SendHello(void);

		int command(int, const char* const*);

		u_int8_t configured; // Flag indicating whether the node has been configured
		nsaddr_t index; // IP address of this node
		int bid; // Broadcast ID

		u_int32_t DynamicDAD_IPAddress; // Address obtained from Wise DAD Allocation

		vector <int> AllocationTable;

		int ReversePath[100][2]; // used for replying
		int ActiveNodes; // number of nodes in the Allocation Table
		int HopCount; // Path Length travelled by a message
		int DAD_trials;      // Number of DAD trials made by an initiator
		int busy;
		int myID;
		u_int32_t DynamicDAD_nid;
		u_int32_t RequestedIP;

		DynamicDADTimer BroadcastTimer; // Timer for AREQ
		DynamicDAD_MembershipReqTimer MembershipReqTimer; // Time for Request to Join
message

		double time, interval;      // Timestamp used in statistics
		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;

		int nodeStatus;
		int MobilityStatus;

		int trafficStatus

		int SizeOfK;
		int Neighnourhood_K [SizeOfK];
		int Min_K_Value;
		int K_NodeStatus = 0 ;
		int mergingStatus ;
		int T; // K Threshold
		int K_Misssing;  //time period W
```

```
            int networkPartitioning_Status;
            int networkMerging_Status;

};

#endif
```

Packet Header File

```
#ifndef _DynamicDAD_PACKET_H_
#define _DynamicDAD_PACKET_H_

#include <packet.h>

#define DynamicDAD_MembershipRequest 0x01 //new node sends this message
#define DynamicDAD_CONFIRMATION 0x02  // a node responding to a Request to join
#define DynamicDAD_InitiatorSelection 0x03 // node selecting an initiator
#define DynamicDAD_AddressRequest 0x04 //initiator sends Areq to detect conflict
#define DynamicDAD_AddressReply   0x05  // initiator replies with this message
#define DynamicDAD_AddressConflict 0x06 // node reporting conflict
#define DynamicDAD_NetworkDeparture 0x07 // node wishing to depart from the network
#define DynamicDAD_ChangeID 0x08 // change ID after partition
#define DynamicDAD_Reset 0x09 // reset config parameters
#define DynamicDAD_sendQ 0x10

// DynamicDAD packet header

struct hdr_DynamicDAD
{
       u_int16_t index; // Initiator of allocation
       u_int16_t DynamicDAD_type; // The type of the packet: State Request, State Reply or
Ack etc
       u_int16_t bcast_id; // Broadcast ID
       u_int32_t DynamicDAD_nid; // Network ID
       u_int32_t DynamicDAD_IPAddress; // this node's IP address
       int counter;
       vector <int> AllocationTable; // IP address Allocation table of thin node

       int ReversePath[100][2]; // Temporary storage of message reverse path
       vector <int> RP;
       vector <RP> ReversePath;
       int HopCount; // number of hops travelled by a message
        u_int32_t DynamicDAD_Request; // requested IP address

       static int offset_; // Required by PacketHeaderManager
       inline static int& offset()
      {
       return offset_;
      }
       inline static hdr_DynamicDAD* access(const Packet* p)
      {
         return (hdr_DynamicDAD*) p->access(offset_);
      }

     inline int size()
     {
      int sz = 0;
      sz = 5*sizeof(u_int32_t);
      return sz;
     }
};
```

```
#define HDR_DynamicDAD(p) ((struct hdr_DynamicDAD*)hdr_DynamicDAD::access(p))

#endif
```

C++ Source File

```
#include "DynamicDAD_packet.h"
#include "DynamicDAD.h"

int hdr_DynamicDAD::offset_;
int i , n ;
int found;
int foundAt;

// Packet Header Class for DynamicDAD address allocation
static class DynamicDADHeaderClass : public PacketHeaderClass
{
       public:
              DynamicDADHeaderClass() : PacketHeaderClass("PacketHeader/DynamicDADHeader",
sizeof(hdr_DynamicDAD))
          {
              bind_offset(&hdr_DynamicDAD::offset_);
          }
} class_DynamicDADhdr;


static class DynamicDADClass : public TclClass {
       public:
              DynamicDADClass() : TclClass("
t/DynamicDAD") { }
              TclObject* create(int argc, const char* const* argv)
              {
                     return (new DynamicDAD);
              }
} class_DynamicDAD;

// Tcl interface for DynamicDAD Allocation
int DynamicDAD::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) {
      time=CURRENT_TIME;
      sendMembershipRequest();
      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])) ih->saddr() = index;== 0) {
```

```
       if (debug == 1) printf( "%s lookup of %s failed\n", argv[1], argv[2]);
       return (TCL_ERROR);
      }
      ll = obj;ih->saddr() = index;
      return (TCL_OK);
    }
  }
  return Agent::command(argc, argv);
}


DynamicDAD::DynamicDAD(void)  :  Agent(PT_DynamicDAD),  btimer(this)  ,  RTJ_Timer(this)
PartitionTimer(this)
{
  configured = 0;
  //bid = 0;
  retries = MAX_RETRY;
  interval = RETRY_TIMEOUT;
  receives = 0;
  DynamicDAD_seq = 0;
  //repeats = 0;

  //bind("bid", &bid);
  //bind("retries", &retries);
  bind("time", &time);
  bind("receives", &receives);
  bind("interval", &interval);
  //bind("seq", &pa_seq);
  bind("debug", &debug);
  //bind("repeats", &repeats);
}

void DynamicDADTimer::expire(Event*)
{
  agent->SendAddressReply();
}


void TempPartitiontimer :: expire(Event*)
{
 agent->InitiatePartition();
}

void DynamicDAD_RTJTimer::expire(Event*)
{
  //agent->committed = 1;
  //Research how "committed" works....
  agent->sendMembershipRequest();
}


void PartitionTimer :: expire(Event*)
{
agent->sendQMessage();

}
```

```
void DynamicDAD::  NetworkTraffic
{
	struct hdr_cmn* ch = HDR_CMN(p);
	struct hdr_ip* ih = HDR_IP(p);
	struct hdr_DynamicDAD* ah = HDR_DynamicDAD(p);
}

void DynamicDAD:: NetworkPartition
{
	struct hdr_cmn* ch = HDR_CMN(p);
	struct hdr_ip* ih = HDR_IP(p);
	struct hdr_DynamicDAD* ah = HDR_DynamicDAD(p);

	int K = neighbours

	if (K < Min_K_Value)
	{


Status = 1;
		PartitionTimer.cancel();
	}

	if (K > Min_K_Value)
	{
		mergingStatus = 0;

	}

	if (K = Min_K_Value)
	{
		mergingStatus = -1;
		PartitionTimer.resched(500);

	}




}


void DynamicDAD:: MobilityStatus
{
	struct hdr_cmn* ch = HDR_CMN(p);
	struct hdr_ip* ih = HDR_IP(p);
	struct hdr_DynamicDAD* ah = HDR_DynamicDAD(p);
}


void DynamicDAD :: TrafficVolume
{
	struct hdr_cmn* ch = HDR_CMN(p);
	struct hdr_ip* ih = HDR_IP(p);
	struct hdr_DynamicDAD* ah = HDR_DynamicDAD(p);
}
```

```
void DynamicDAD::recv(Packet* p, Handler*)
{
  struct hdr_cmn* ch = HDR_CMN(p);
  struct hdr_ip* ih = HDR_IP(p);
  struct hdr_DynamicDAD* ah = HDR_DynamicDAD(p);

  // put all  variables received here.
  // use the variables when replying.
  nsaddr_t src = ih->saddr();
  HopCount =  ah->HopCount;
  RequestedIP = ah-> DynamicDAD_Request;

  for (i=0; i<=99; i++)
  {
   ReversePath[i][0] = ah->ReversePath[i][0];
  }

  for (i=0; i<=99; i++)
  {
   ReversePath[i][1] = ah->ReversePath[i][1];
  }

  switch(ah->DynamicDAD_type) {

case DynamicDAD_MembershipRequest:
       // a node can only be an inititor of one node at a time
       // If a node is busy, it doesnt respond to Request To Join
       // And also if a node is not configured,  it doestn respond to a RTJ
       if (configured !=0 && busy == 0)
              {
                     sendCONFIRMATION(src);
                     receives++;
              }

       Packet::free(p);
       break;


case DynamicDAD_CONFIRMATION:
       //
       if (configured == 0 && receives == 0)
              {
                     sendSELECT_INITIATOR(scr);
                     receives++;

              }
       Packet::free(p);
       break;

case DynamicDAD_AddressReply:
    if (configured == 0)
       {
        // a new node configures itself after a successful DynamicDAD
        DynamicDAD_IPAddress = ah->DynamicDAD_Request;
        DynamicDAD_nid = ah->DynamicDAD_nid;
        AllocationTable  = ah->AllocationTable;
```

```
        RTJ_Timer.cancel();
       }
    Packet::free(p);
    break;


case DynamicDAD_AddressConflict:
        if (ah->HopCount != 0)
       {
          send_IP_CONFICT(ah->ReversePath[HopCount][0] - 1);
       }

       if (ah->HopCount == 0)
       {
           // Generate another IP address and perform a local DAD before AddressRequest
            counter = 0;
           found = 0;
            while (counter < AllocationTable.size() || found == 0 )
           {
             counter = counter + 1;
             RequestedIP = (Random::integer(65536));
             found  = 0;
             for (i=0; i<= ActiveNodes; i++)
              {
                 if (AllocationTable[i] == RequstedIP)
                   {
                     found = 1;
                   }
              }
             }

             HopCount = 0;
            DAD_trials = DAD_trials + 1;
             SendAddressRequest();
       }
       Packet::free(p);
       break;

case DynamicDAD_AddressRequest:
       //checking if this node has received this message before
       // and discarding th message
       found = 0;
       for (i=0; i<=ah->HopCount; i++)
       {
              if (ah->ReversePath[i][1] == DynamicDAD_IPAddress)
              {
                found = 1;
              }
       }
       if (found == 1)
       {
         Packet::free(p);
       }

// if message is new then process it.
if (found == 0)

{      //check for conflict then send an IPConflict message
```

```
if (DynamicDAD_IPAddress == ah->DynamicDAD_Request)
{

       sendIP_CONFLICT(ah->ReversePath[HopCount][1] - 1);
 }

//update allocation table ie move the requested address up
if (DynamicDAD_IPAddress != ah->DynamicDAD_Request)
{ found = 0;
 for (i = 0; i<=ActiveNodes; i++)
       {
         if (ah->DynamicDAD_Request == AllocationTable[i])
              {
                found = 1;
                foundAt = i;
              }
       }
 }

if (found == 1)
 {
     AllocationTable.erase(foundAt + 1)
     AllocationTable.push_back(ah->DynamicDAD_Request);
 }

if (found  == 0)
 {
    AllocationTable.push_back(ah->WiseDAah->ReversePath[i][1]);
 }

//checking Reverse path and updating Allocation table

int j;
ActiveNodes = AllocationTable.size();
for (i = 0; i>= ah->HopCount; i++)
       {
           found = 0;
         for (j=0; j<= ActiveNodes; j++)
         if (ah->ReversePath[i][1] == AllocationTable[j])
              {
                AllocationTable.erase(i + 1)
                AllocationTable.push_back(ah->ah->ReversePath[i][1]);
                found  = 1;
              }
         if (found == 0)
         {
              AllocationTable.push_back(ah->ReversePath[i][1]);
         {

       }

// forwarding AddressRequest
       AllocAllocationTable.push_backationTable.push_back
if (DynamicDAD_IPAddress != ah->DynamicDAD_Request)
{

       SendAddressRequest();
 }
```

```
        Packet::free(p);
}
        break;

case DynamicDAD_InitiatorSelection:
        //try and check if message is mine
        // there is need to check allocation table first before AddressRequest (Local DAD)
         counter = 0;
          found = 0;
           while (counter < AllocationTable.size() || found == 0 )
          {
           counter = counter + 1;
           RequestedIP = (Random::integer(65536)+1);
           found  = 0;
            for (i=0; i<= ActiveNodes; i++)
               {
                  if (AllocationTable[i] == RequstedIP)
                    {
                      found = 1;
                    }
               }
          }
        HopCount = 0;
        DAD_trials = 1;
        SendAddressRequest();

       Packet::free(p);
        break;

case DynamicDAD_NetworkDeparture:
        //checking if this node has received this message before

         found = 0;
        for (i=0; i<=ah->HopCount; i++)
        {
               if (ah->ReversePath[i][1] == DynamicDAD_IPAddress)
               {
                 found = 1;
               }
        }
        Packet::free(p);

//if message is new, process it
if (found == 0)
{
        //checking if the departing node is known then delete it
        found = 0;
         for (i = 0; i<=ActiveNodes; i++)
               {
                 if ah->DynamicDAD_Request == Allocation[i]
                      {
                        found = 1;
                        foundAt = i;
                      }
               }
          // remove the IP from the allocation table
         if (found == 1)
         {
```

```
         AllocationTable.erase(foundAt + 1);
        }
       // updating Allocation table using Reverse path
       int j;e launching a project to improve security for the great apes.
       ActiveNodes = AllocationTable.size();
       for (i = 0; i>= ah->HopCount; i++)
              {
                  found = 0;
             for (j=0; j<= ActiveNodes; j++)
             if (ah->ReversePath[i][1] == AllocationTable[j])
                  {
                    AllocationTable.erase(i + 1)
                    AllocationTable.push_back(ah->ah->ReversePath[i][1]);
                    found  = 1;
                  }
             if (found == 0)
             {
                  AllocationTable.push_back(ah->ReversePath[i][1]);
             {

              }

       sendGOODBYE();
       Packet::free(p);
}
       break;

  }
}

void DynamicDAD::sendMembershipRequest(void)
{
 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_DynamicDAD* ah = HDR_DynamicDAD(p);
       ch->size() = IP_HDR_LEN + ah->size();

        ih->daddr() = IP_BROADCAST;
        ih->saddr() = index;
        ih->ttl_() = 1;

        ah->DynamicDAD_type = DynamicDAD_MembershipRequest;

        if (debug == 1) {printf("Node %d broadcasts a request packet\n", index);}
        Scheduler::instance().schedule(target_, p, 0.0);
       // request should be rescheduled if no reply is received after a certain interval.
   }
  RTJ_Timer.resched(interval);

 // Here, the first node configures itself and chooses the network parameters (Network ID)
  if (index == 0)
   {
      DynamicDAD_index = 0;

      DynamicDAD_nid = (Random::integer(65536));
```

```
      configured = 1;
      DynamicDAD_IPAddress = (Random::integer(65536));
      printf("First Node chooses %d as its IP address and as the Network ID %d \n",
DynamicDAD_IPAddress, DynamicDAD_nid);
      AllocationTable.push_back(DynamicDAD_IPAddress);
   }
}

void DynamicDAD::sendCONFIRMATION(nsaddr_t dest)
{
  Packet* p = Packet::alloc();
  struct hdr_cmn* ch = HDR_CMN(p);
  struct hdr_ip* ih = HDR_IP(p);
  struct hdr_DynamicDAD* ah = HDR_DynamicDAD(p);
  ch->size() = IP_HDR_LEN + ah->size();

  ih->daddr() = dest; //  Chosen initiator's address
  ih->saddr() = index;  // source of this message

  ah->DynamicDAD_type = DynamicDAD_CONFIRMATION;
  ah->DynamicDAD_nid = 0;
  ah->DynamicDAD_IPAddress = 0;

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

void void DynamicDAD::ResetPartition(void)
{
Packet* p = Packet::alloc();
  struct hdr_cmn* ch = HDR_CMN(p);
  struct hdr_ip* ih = HDR_IP(p);
  struct hdr_DynamicDAD* ah = HDR_DynamicDAD(p);
  ch->size() = IP_HDR_LEN + ah->size();
this->Min_K_Value = 0;
this->K_NodeStatus = 0 ;
this->mergingStatus = 0;

}


void DynamicDAD::OldMessage(void)
{

       Packet* p = Packet::alloc();
       found = 0;
       for (i=0; i<=ah->HopCount; i++)
       {
              if (ah->ReversePath[i][1] == DynamicDAD_IPAddress)
              {
                found = 1;
              }
       }
       if (found == 1)
       {
         Packet::free(p);
       }
}
```

```
void DynamicDAD::DetPartition(void)
{
  Packet* p = Packet::alloc();
  struct hdr_cmn* ch = HDR_CMN(p);
  struct hdr_ip* ih = HDR_IP(p);
  struct hdr_DynamicDAD* ah = HDR_DynamicDAD(p);
  ch->size() = IP_HDR_LEN + ah->size();
  ih->daddr() = IP_BROADCAST;
  ih->saddr() = index;
  ah->DynamicDAD_type = DynamicDAD_Reset;
  Scheduler::instance().schedule(target_, p, 0.0);
}


void DynamicDAD::InitiatePartition(void)

{
Packet* p = Packet::alloc();
  struct hdr_cmn* ch = HDR_CMN(p);
  struct hdr_ip* ih = HDR_IP(p);
  struct hdr_DynamicDAD* ah = HDR_DynamicDAD(p);
  ch->size() = IP_HDR_LEN + ah->size();

  ih->daddr() = IP_BROADCAST;
  ih->saddr() = index;

  generateID();
 ah->HopCount = HopCount + 1;

if (HopCount == 0)
  {
   btimer.resched(interval);
   busy = 1;
ih->ID = myID;
  }

  ah->DynamicDAD_type = DynamicDAD_ChangeID;
  Scheduler::instance().schedule(target_, p, 0.0);
}

void DynamicDAD::SendAddressRequest(void)
{
  Packet* p = Packet::alloc();
  struct hdr_cmn* ch = HDR_CMN(p);
  struct hdr_ip* ih = HDR_IP(p);
  struct hdr_DynamicDAD* ah = HDR_DynamicDAD(p);
  ch->size() = IP_HDR_LEN + ah->size();

  ih->daddr() = IP_BROADCAST;
  ih->saddr() = index;

  ah->DynamicDAD_type = DynamicDAD_AddressRequest;
  ah->DynamicDAD_Request = RequestedIP;
  // Adding my own IP and my own DynamicDAD to reverse Path
  // My IP will be stored at position [hopcount] on the reverse path vector
  ah->ReversePath[HopCount][0] = DynamicDAD_IPAddress;
  ah->ReversePath[HopCount][1] = ih->saddr();
  // increase hop count then send message
```

```
  ah->HopCount = HopCount + 1;
  // if the sender of AddressRequest is the source of the message, then it has to reschedule
the timer
if (HopCount == 0)
  {
   btimer.resched(interval);
   busy = 1;
  }

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

void DynamicDAD::sendIPConflict(nsaddr_t dest)
{
  Packet* p = Packet::alloc();
  struct hdr_cmn* ch = HDR_CMN(p);
  struct hdr_ip* ih = HDR_IP(p);
  struct hdr_DynamicDAD* ah = HDR_DynamicDAD(p);
   ch->size() = IP_HDR_LEN + ah->size();

  //ah->ReversePath[ah->HopCount][0] = DynamicDAD_IPAddress;
  //ah->ReversePath[ah->HopCount][1] = ih->saddr();
  ah->HopCount = HopCount - 1;

  ih->daddr() = dest;
  ih->saddr() = index;

  ah->DynamicDAD_type = DynamicDAD_IP_CONFLICT;
  ah->DynamicDAD_Request = RequestedIP;

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

void DynamicDAD::SendAddressReply(nsaddr_t dest)
{
  Packet* p = Packet::alloc();
  struct hdr_cmn* ch = HDR_CMN(p);
  struct hdr_ip* ih = HDR_IP(p);
  struct hdr_DynamicDAD* ah = HDR_DynamicDAD(p);
  ch->size() = IP_HDR_LEN + ah->size();

  ih->daddr() = IP_BROADCAST;
  ih->saddr() = index;

  ih->ttl_ = 1;
  ah->DynamicDAD_type = DynamicDAD_AREP;
  ah->DynamicDAD_Request = RequestedIP;
  ah->DynamicDAD_nid = DynamicDAD_nid;
  ah->AllocationTable = AllocationTable;

  btimer.cancel();

//  busy = 0;

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

void DynamicDAD::SendAddressReply(nsaddr_t dest)
```

```
{
  Packet* p = Packet::alloc();
  struct hdr_cmn* ch = HDR_CMN(p);
  struct hdr_ip* ih = HDR_IP(p);
  struct hdr_DynamicDAD* ah = HDR_DynamicDAD(p);
  ch->size() = IP_HDR_LEN + ah->size();

  ih->daddr() = IP_BROADCAST;
  ih->saddr() = index;

  ih->ttl_ = 1;
  ah->DynamicDAD_type = DynamicDAD_sendQ;
  ah->DynamicDAD_Request = RequestedIP;
  ah->DynamicDAD_nid = DynamicDAD_nid;


  btimer.cancel();

//  busy = 0;

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

void DynamicDAD::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_DynamicDAD* ah = HDR_DynamicDAD(p);
  ch->size() = IP_HDR_LEN + ah->size();

  ih->daddr() = dest;
  ih->saddr() = index;DynamicDAD_nid;

  ah->DynamicDAD_type = DynamicDAD_SELECT_INITIATOR;
  ah->DynamicDAD_Request = 0;
  ah->DynamicDAD_nid = 0;

  printf("Node %d sent a SELECT INIT to node %d with IP address %d\n", index, ih->daddr(),
DynamicDAD_IPAddress);

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

void DynamicDAD::sendGOODBYE(void)
{
  Packet* p = Packet::alloc();
  struct hdr_cmn* ch = HDR_CMN(p);
  struct hdr_ip* ih = HDR_IP(p);
  struct hdr_DynamicDAD* ah = HDR_DynamicDAD(p);
  ch->size() = IP_HDR_LEN + ah->size();

  ih->daddr() = IP_BROADCAST;
  ih->saddr() = index;

  ah->DynamicDAD_type = DynamicDAD_GOODBYE;
  ah->DynamicDAD_Request = RequestedIP;
  ah->DynamicDAD_nid  = DynamicDAD_nid;
```

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

TCL Sample Code for Testing the Protocol

```
#===================================================================
# 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)      50                       ; # number of mobile nodes
set val(x)       500
set val(y)       500
#set val(seed)    0.0
set val(sc)      scen-50-500x500
set val(stop)    1600.0                   ; # simulation time
set val(god)     off
set val(intv)    10
set val(k)       3
set val(t)       [expr $val(intv) * $val(k)]
set ns_ [new Simulator]
```

```
$ns_ use-scheduler Heap

set tracefd [open DynamicDAD.tr w]
$ns_ trace-all $tracefd

set namtrace [open DynamicDAD-out.nam w]     ; # for wireless traces
$ns_ namtrace-all-wireless $namtrace $val(x) $val(y)

set DataFile0 [open DynamicDAD-out0.tr w]
set DataFile1 [open DynamicDAD-out1.tr w]
set DataFile2 [open DynamicDAD-out2.tr w]


proc finish {} {
    global DataFile0 DataFile1 DataFile2
    close $DataFile0
    close $DataFile1
    close $DataFile2
    exec xgraph DynamicDAD-out0.tr DynamicDAD-out1.tr &
    exit 0
}

proc record { DynamicDAD } {
    global DataFile0 DataFile1 DataFile2
    set index [$DynamicDAD id]
```

```
    set recvs [$DynamicDAD set receives]
    set retrs [$DynamicDAD set repeats]
    #set random [$DynamicDAD set seq]
    puts $DataFile0 "$index $recvs"
    puts $DataFile1 "$index $retrs"
    puts $DataFile2 "$index $random"
}

#
# Define topology
#
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)

#
# Create GoD
#
set god_ [create-god $val(nn)]
#$god_ $val(god)
#$god_ allow_to_stop
#$god_ num_data_types 1

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)
}

#
# Define node movement model
#
puts "Loading scenario file..."
```

```
source $val(sc)

#
# Define initial position in nam
#
for { set i 0 } { $i < $val(nn) } { incr i } {
   $ns_ initial_node_pos $node_($i) 50
}

# Setup a the DynamicDAD agent
for { set i 0 } { $i < $val(nn) } { incr i } {
   set DynamicDAD_($i) [new Agent/PA $i]
   $DynamicDAD_($i) index $i
   $DynamicDAD_($i) set interval $val(intv)
   $DynamicDAD_($i) set retries $val(k)
   #$DynamicDAD_($i) set debug 1
   $ns_ attach-agent $node_($i) $DynamicDAD_($i)
   set ll($i) [$node_($i) set ll_(0)]
   $DynamicDAD_($i) set-ll $ll($i)
}

$ns_ at [expr 10.0+[$DynamicDAD_(0) id]*$val(t)] "$DynamicDAD_(0) start"

$ns_ at [expr 10.0+[$DynamicDAD_(1) id]*$val(t)] "$DynamicDAD_(1) start"

$ns_ at [expr 10.0+[$DynamicDAD_(2) id]*$val(t)] "$DynamicDAD_(2) start"
```

```
$ns_ at [expr 10.0+[$DynamicDAD_(3) id]*$val(t)] "$DynamicDAD_(3) start"

$ns_ at [expr 10.0+[$DynamicDAD_(4) id]*$val(t)] "$DynamicDAD_(4) start"

$ns_ at [expr 10.0+[$DynamicDAD_(5) id]*$val(t)] "$DynamicDAD_(5) start"

$ns_ at [expr 10.0+[$DynamicDAD_(6) id]*$val(t)] "$DynamicDAD_(6) start"

$ns_ at [expr 10.0+[$DynamicDAD_(7) id]*$val(t)] "$DynamicDAD_(7) start"

$ns_ at [expr 10.0+[$DynamicDAD_(8) id]*$val(t)] "$DynamicDAD_(8) start"

$ns_ at [expr 10.0+[$DynamicDAD_(9) id]*$val(t)] "$DynamicDAD_(9) start"

$ns_ at [expr 10.0+[$DynamicDAD_(10) id]*$val(t)] "$DynamicDAD_(10) start"

$ns_ at [expr 10.0+[$DynamicDAD_(11) id]*$val(t)] "$DynamicDAD_(11) start"

$ns_ at [expr 10.0+[$DynamicDAD_(12) id]*$val(t)] "$DynamicDAD_(12) start"

$ns_ at [expr 10.0+[$DynamicDAD_(13) id]*$val(t)] "$DynamicDAD_(13) start"

$ns_ at [expr 10.0+[$DynamicDAD_(14) id]*$val(t)] "$DynamicDAD_(14) start"

$ns_ at [expr 10.0+[$DynamicDAD_(15) id]*$val(t)] "$DynamicDAD_(15) start"
```

$ns_ at [expr 10.0+[$DynamicDAD_(16) id]*$val(t)] "$DynamicDAD_(16) start"

$ns_ at [expr 10.0+[$DynamicDAD_(17) id]*$val(t)] "$DynamicDAD_(17) start"

$ns_ at [expr 10.0+[$DynamicDAD_(18) id]*$val(t)] "$DynamicDAD_(18) start"

$ns_ at [expr 10.0+[$DynamicDAD_(19) id]*$val(t)] "$DynamicDAD_(19) start"

$ns_ at [expr 10.0+[$DynamicDAD_(20) id]*$val(t)] "$DynamicDAD_(20) start"

$ns_ at [expr 10.0+[$DynamicDAD_(21) id]*$val(t)] "$DynamicDAD_(21) start"

$ns_ at [expr 10.0+[$DynamicDAD_(22) id]*$val(t)] "$DynamicDAD_(22) start"

$ns_ at [expr 10.0+[$DynamicDAD_(23) id]*$val(t)] "$DynamicDAD_(23) start"

$ns_ at [expr 10.0+[$DynamicDAD_(24) id]*$val(t)] "$DynamicDAD_(24) start"

$ns_ at [expr 10.0+[$DynamicDAD_(25) id]*$val(t)] "$DynamicDAD_(25) start"

$ns_ at [expr 10.0+[$DynamicDAD_(26) id]*$val(t)] "$DynamicDAD_(26) start"

$ns_ at [expr 10.0+[$DynamicDAD_(27) id]*$val(t)] "$DynamicDAD_(27) start"

$ns_ at [expr 10.0+[$DynamicDAD_(28) id]*$val(t)] "$DynamicDAD_(28) start"

$ns_ at [expr 10.0+[$DynamicDAD_(29) id]*$val(t)] "$DynamicDAD_(29) start"

$ns_ at [expr 10.0+[$DynamicDAD_(30) id]*$val(t)] "$DynamicDAD_(30) start"

$ns_ at [expr 10.0+[$DynamicDAD_(31) id]*$val(t)] "$DynamicDAD_(31) start"

$ns_ at [expr 10.0+[$DynamicDAD_(32) id]*$val(t)] "$DynamicDAD_(32) start"

$ns_ at [expr 10.0+[$DynamicDAD_(33) id]*$val(t)] "$DynamicDAD_(33) start"

$ns_ at [expr 10.0+[$DynamicDAD_(34) id]*$val(t)] "$DynamicDAD_(34) start"

$ns_ at [expr 10.0+[$DynamicDAD_(35) id]*$val(t)] "$DynamicDAD_(35) start"

$ns_ at [expr 10.0+[$DynamicDAD_(36) id]*$val(t)] "$DynamicDAD_(36) start"

$ns_ at [expr 10.0+[$DynamicDAD_(37) id]*$val(t)] "$DynamicDAD_(37) start"

$ns_ at [expr 10.0+[$DynamicDAD_(38) id]*$val(t)] "$DynamicDAD_(38) start"

$ns_ at [expr 10.0+[$DynamicDAD_(39) id]*$val(t)] "$DynamicDAD_(39) start"

$ns_ at [expr 10.0+[$DynamicDAD_(40) id]*$val(t)] "$DynamicDAD_(40) start"

$ns_ at [expr 10.0+[$DynamicDAD_(41) id]*$val(t)] "$DynamicDAD_(41) start"

```
$ns_ at [expr 10.0+[$DynamicDAD_(42) id]*$val(t)] "$DynamicDAD_(42) start"

$ns_ at [expr 10.0+[$DynamicDAD_(43) id]*$val(t)] "$DynamicDAD_(43) start"

$ns_ at [expr 10.0+[$DynamicDAD_(44) id]*$val(t)] "$DynamicDAD_(44) start"

$ns_ at [expr 10.0+[$DynamicDAD_(45) id]*$val(t)] "$DynamicDAD_(45) start"

$ns_ at [expr 10.0+[$DynamicDAD_(46) id]*$val(t)] "$DynamicDAD_(46) start"

$ns_ at [expr 10.0+[$DynamicDAD_(47) id]*$val(t)] "$DynamicDAD_(47) start"

$ns_ at [expr 10.0+[$DynamicDAD_(48) id]*$val(t)] "$DynamicDAD_(48) start"

$ns_ at [expr 10.0+[$DynamicDAD_(49) id]*$val(t)] "$DynamicDAD_(49) start"



#
# Tell nodes when the simulation ends
#

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

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

puts "Starting simulation..."
$ns_ run
```

“Only the dead have seen the end of war”