RMIT UNIVERSITY

Artificial Immune System based Security Algorithm for Mobile Ad Hoc Networks

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

Lincy Elizebeth Jim

M.Tech in Electronics Design and Technology, National Institute of Technology, Calicut, India

B.Tech in Electronics and Communication, Cochin University of Science and Technology, India

School of Engineering

College of Science, Engineering and Health

RMIT University

August 2017

RMIT UNIVERSITY

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; any editorial work, paid or unpaid, carried out by a third party is acknowledged; and, ethics procedures and guidelines have been followed.

I acknowledge the support I have received for my research through the provision of an Australian Government Research Training Program Scholarship

The work presented within this thesis holds no material or information that has been accepted for the award in any university for any degree. To the best of my knowledge and belief, this thesis does not contain any material written by another person except for the places denoted by specific references. The content of this thesis is the product of research work carried out at RMIT University, since the start of this research program.

Lincy Elizebeth Jim

31/08/2017

RMIT UNIVERSITY

My PhD

It has been a constant battle, a battle where fears and emotions try to take advantage of you, "Do not give up " the voice says, "For the seeds you sow today in your life will make great trees that will provide you comfort later!!"

It's not everyone who have tread this path and for those trodden this path a huge applause for all your hard work, dedication and patience!!!!

Always remember the darkest hour is always before dawn!

"This is dedicated to my God who kept me moving forward with perseverance"

My appreciation and thanks go to:

My husband, kids and my parents,

For their love, endless support and sacrifice

RMIT UNIVERSITY

Acknowledgements

I would like to acknowledge the financial, academic and technical support of the School of Engineering, RMIT University, and its staff. I would like to acknowledge the opportunity given to me by my primary supervisor, Associate Professor Mark A Gregory, in order to undertake this PhD. He undoubtedly had the most profound influence on the achievements during my PhD research. On that note, I would like to thank him for showing patient support, constant encouragement, and assistance throughout my research program. Most importantly, I am thankful to him for easing me into this program and for trusting my capabilities during the research tenure. I deeply acknowledge Professor Henry Wu for his engagement in many constructive discussions over email and critical reviews on my model prior to finalize the result. I also appreciate the many anonymous reviewers of my peer-reviewed published journal and conference proceedings. Their detailed, insightful and sometimes tough comments helped me to look at every fine detail of the project that improve the quality of the work immeasurably.

I would like to extend a special note of thanks to my dearest husband Mr. Jim Chacko for his tremendous patience and support during the entire stretch of this PhD program. I would also like to thank my parents for providing their constant support throughout my life. To my little children Johna Susan Jim and Jaiden Jim, who have been affected in every way possible by this quest and helped me maintain straight thinking with their innocence. Thank you very much for your valuable prayers. My love for the both of you darlings can never be measured. God bless you. Finally, for moral support I extend many thanks to my beloved father, Mr. Elavinamannil Thomas Kuriakose and my mother, Mrs. Lisy Kuriakose as well as my brother

Mr. Lejoe Thomas Kuriakose and sister in law Dr. Biji Lejoe, as well as my dearest friends Mr. Peter Paily and Ms. Ann Manoj for which my mere expression of thanks likewise does not suffice.

RMIT UNIVERSITY

Abstract

Securing Mobile Ad hoc Networks (MANET) that are a collection of mobile, decentralized, and self-organized nodes is a challenging task. The most fundamental aspect of a MANET is its lack of infrastructure, and most design issues and challenges stem from this characteristic. The lack of a centralized control mechanism brings added difficulty in fault detection and correction. The dynamically changing nature of mobile nodes causes the formation of an unpredictable topology. This varying topology causes frequent traffic routing changes, network partitioning and packet losses. The various attacks that can be carried out on MANETs challenge the security capabilities of the mobile wireless network in which nodes can join, leave and move dynamically. The Human Immune System (HIS) provides a foundation upon which Artificial Immune algorithms are based. The algorithms can be used to secure both host-based and network-based systems. However, it is not only important to utilize the HIS during the development of Artificial Immune System (AIS) based algorithms as much as it is important to introduce an algorithm with high performance. Therefore, creating a balance between utilizing HIS and AIS-based intrusion detection algorithms is a crucial issue that is important to investigate.

The immune system is a key to the defence of a host against foreign objects or pathogens. Proper functioning of the immune system is necessary to maintain host homeostasis. The cells that play a fundamental role in this defence process are known as Dendritic Cells (DC). The AIS based Dendritic Cell Algorithm is widely known for its large number of applications and well established in the literature.

The dynamic, distributed topology of a MANET provides many challenges, including

decentralized infrastructure wherein each node can act as a host, router and relay for traffic. MANETs are a suitable solution for distributed regional, military and emergency networks. MANETs do not utilize fixed infrastructure except where a connection to a carrier network is required, and MANET nodes provide the transmission capability to receive, transmit and route traffic from a sender node to the destination node. In the HIS, cells can distinguish between a range of issues including foreign body attacks as well as cellular senescence.

The primary purpose of this research is to improve the security of MANET using the AIS framework. This research presents a new defence approach using AIS which mimics the strategy of the HIS combined with Danger Theory. The proposed framework is known as the Artificial Immune System based Security Algorithm (AISBA).

This research also modelled participating nodes as a DC and proposed various signals to indicate the MANET communications state. Two trust models were introduced based on AIS signals and effective communication. The trust models proposed in this research helped to distinguish between a “good node” as well as a “selfish node”.

A new MANET security attack was identified titled the Packet Storage Time attack wherein the attacker node modifies its queue time to make the packets stay longer than necessary and then circulates stale packets in the network. This attack is detected using the proposed AISBA.

This research, performed extensive simulations with results to support the effectiveness of the proposed framework, and statistical analysis was done which showed the false positive and false negative probability falls below 5%.

Finally, two variations of the AISBA were proposed and investigated, including the Grudger based Artificial Immune System Algorithm - to stimulate selfish nodes to cooperate for the benefit of the MANET and Pain reduction based Artificial Immune System Algorithm - to model Pain analogous to HIS.

RMIT UNIVERSITY

Table of Contents

RMIT UNIVERSITY

List of Figures

RMIT
UNIVERSITY

RMIT UNIVERSITY

List of Tables

RMIT UNIVERSITY

List of Acronyms and Abbreviations

AIS	Artificial Immune System
AISBA	Artificial Immune System Based Security Algorithm
DC	Dendritic Cell
DS	Danger Signal
GrAIS	Grudger Based Artificial Immune System Algorithm
HIS	Human Immune System
MANET	Mobile Ad hoc Network
PST	Packet Storage Time Attack
PAMP	Pathogen Associated Molecular Pattern
PrAIS	Pain Reduction Artificial Immune System Algorithm
SS	Safe Signal
IS	Identifier Signal
SC	Secure Signal
WL	WorkLoad

Chapter 1 INTRODUCTION

1.1 Introduction

Wireless communications is a key technology that permits consumers to participate in the global digital economy as they go about their daily activities. Fixed and mobile wireless communications have evolved rapidly as demand for telecommunications has increased and today the rate of technological change is happening faster than ever before. Mobile ad hoc networks (MANETs) are a form of wireless communications that consist of an assemblage of wireless mobile nodes which dynamically interchange data amongst themselves without the dependence on a fixed base station or a wired backbone network. The goal of MANET is to sustain reliable mobile communications by amalgamating the network routing and relay functionality into the mobile nodes. The limited transmission range of the mobile nodes means that multiple hops could be required to exchange information with other nodes [2]. During the last decade, studies have been carried out on MANET routing that have resulted in numerous mature routing protocols. However, in most cases, the protocols require a trusted MANET environment, which is difficult to achieve. In many situations, the environment is susceptible to a range of security and other issues. For example, various behavioural patterns can be exhibited in nodes; some nodes can be selfish, malicious, or compromised by attackers. Various strategies have been advocated to secure the MANET routing protocols to identify and mitigate the various forms of attack, however, MANET security remains an active research topic [3]. In this thesis, a literature review is provided that identifies the MANET security challenges and an Artificial Immune System (AIS) approach is used to secure MANET from selected security related attacks.

The MANET nodes that are in transmission range of each other are called neighbours.

Neighbours should be able to communicate with each other [2,3], however, when a node needs to forward data to other non-neighbouring nodes, the data might be routed through a sequence of multiple hops, with intermediate nodes acting as routers or relays. The success of MANET strongly depends on whether the nodes can be trusted.

1.2 Research Problem

MANETs are evolving and the threat landscape has increased, making the security of MANET a research focus. Mobility provides advantages and the ad hoc network nature of MANETs adds to the advantages, however an insecure MANET would negate many of the advantages. Ad hoc networks may be deployed in various terrain and hazardous conditions and even hostile environments where the device may be compromised, faulty or unserviceable.

- Resource-constrained nodes. MANET nodes are typically battery powered and have limited storage and processing capability. Moreover, they may be situated in areas where it is not possible to re-charge and thus have limited lifetimes. Because of these limitations, they must be well designed to optimize energy-efficient operation within the limits of the storage available and processing capability.

- Dynamic topology. The topology in an ad hoc network may change continuously due to the node mobility. As nodes move in and out of range of each other, some links break while new links between nodes are created. Because of these issues, MANETs are prone to various faults.

- Node failures. Nodes may fail at any time due to different types of hazardous

environmental conditions. They may also drop out of the network either voluntarily or when their energy supply is depleted.

- Link failures. Node faults and changing environmental conditions (e.g., increased EMI) may cause links between nodes to degrade or fail.

- Route failures. When the network topology changes due to node mobility, nodes being added or removed from the network or faults and failures, traffic routes become out of date regularly and quickly. Depending upon the network transport protocol, packets forwarded through stale routes may eventually be dropped or delayed; packets may take a circuitous route before eventually arriving at the destination node. MANET routing protocols should deal with these issues to be effective.

- Dynamic topology. Dynamic topology and changeable node membership may disturb the trust relationship between nodes. The trust may also be disturbed if some of the nodes are detected as compromised. This dynamic behaviour leads to the need for distributed and adaptive security mechanisms.

- Adversary inside the network. The MANET nodes can freely join and leave the network and if they've joined the network there is an anticipation that the nodes will participate in the MANET, however, MANET nodes restrict power usage and this could lead to a selfish node behavior. Selfish behaviour by MANET nodes can lead to severely degraded performance making their behaviour mode detrimental in some respects than an external security attack.

The following research challenges were identified:

- Applying the HIS concept to MANET to design a “bio immune MANET”
- Implementing a DC in a MANET context
- Utilising the HIS concept to address security concerns
- Variations to typical security attacks
- How to overcome selfish node behaviour
- Modelling pain in a MANET

1.3 Research Aims

The research carried out encompassed several of the MANET challenges with a focus on the development of a robust and reliable security framework utilizing AIS concepts. The research aims include:

1. In the current state of the art the nodes are in a protected state or human intervention is required when facing security threats. These situations are impractical in a MANET which is known for dynamic topology and node mobility. Therefore, a new and innovative approach is necessary which can overcome the challenges of the existing MANET design and rectify the drawbacks of the current state of the art. Integration of an AIS scheme in MANET packet transmission in order to create AIS based routing (Translate AIS signals to MANET signals) has the potential to be a valuable framework.

2. Model an AIS based security algorithm. Each node is modelled as a DC that initiate immune responses. Each DC node monitors the routing process and generates signals indicating the presence or absence of danger.

3. Model the Packet Storage Time Attack.

4. Model the Grudger based Artificial Immune System based Algorithm that helps to stimulate the cooperation of selfish nodes.

5. Model Pain Reduction Algorithm utilizing AIS concepts to alleviate pain in MANET

1.4 Objective

The objective of this research was to:

1. Analyse nodes based on their states and investigate the Probability of Communication as this is important when modelling MANET nodes as DCs.

2. Implement an AIS based Security Algorithm (AISBA) and model the algorithm performance using trust metrics and AIS signals.

3. Implement the Packet Storage Time (PST) attack and investigate its effect on MANET.

4. Implement a Grudger based Artificial Immune System Algorithm (GrAIS) and model the algorithm to see the effects of selfish node stimulation to cooperate in packet transmission.

5. Implement a Pain reduction based Artificial Immune System Algorithm (PrAIS) and

model the algorithm to identify MANET Pain analogous to that found in the HIS.

1.5 Research Contribution

The research carried out successfully met the research aims and objectives. An improved AIS based framework for MANET that incorporated solutions for several of the MANET challenges was developed. An AISBA model was proposed and simulated using NS-3. AISBA realizes a high detection rate and packet delivery ratio. The selfish nodes that usually behave in an adversarial manner were stimulated to cooperate based on the GrAIS model. Pain is a stimulus which indicates the cells in HIS are under stress or duress, this concept is the first of its kind, namely that PrAIS was modelled in a MANET. Finally, a new routing attack PST [59] was designed and simulations were carried out to validate the PST attacker utilizing AIS principles.

The research contribution includes:

- A review of the literature and a thematic classification of various AIS algorithms. A classification is proposed according to the challenges that AIS based MANET schemes might attempt to solve, thus providing a more efficient understanding of the proposed solution. In addition, the security attacks in MANET have also been detailed thereby providing an understanding of the reason behind the investigation of a new routing attack in MANET.

- AISBA has been designed with AIS signals to provide a secure routing algorithm to detect selfish nodes. This is inspired from the HIS as the DCs in the Human body provide a robust defence. To guarantee reliability and minimizing end-to-end latency,

Trust metrics have been modelled and utilized to provide secure routing for MANET nodes. Extensive simulations demonstrate that AISBA yields a significant improvement in detection rate and packet delivery ratio.

- A novel routing attack, PST, is implemented in MANET. In PST, the attacker modifies the storage time of the packet so that it does not reach the intended destination nodes. Utilizing AIS signals the source of the PST attack was identified.

- A variant of the AISBA, GrAIS, takes advantage of the idea of a Dawkins model of birds and transforms the issue of selfish nodes non-cooperation by stimulating them to cooperate by utilizing the concept of increasing workload. Simulation results show that GrAIS yields significant improvements in the efficiency of packet delivery.

- The PrAIS approach models pain in MANET. This novel approach is the first of its kind where pain that is analogous to what is found in the HIS is added to a MANET architecture. PrAIS applies a Pain before action (P_{ba}) and Pain after action (P_{aa}) based Pain Reduction approach, which uses the AIS signals, and trust among the nodes. Extensive simulations have demonstrated the efficiency and effectiveness of the proposed approach.

1.6 Publications

Peer reviewed publications either published or submitted for publication during the research program.

Journal

1. L. E. Jim and M. A. Gregory, "A review of artificial immune system based security frameworks for Manet," International Journal of Communications, Network and System Sciences, vol. 9, p. 1, 2016. (PUBLISHED)

2. L. E. Jim and M. A. Gregory, "Utilisation of DANGER and PAMP signals to detect a MANET Packet Storage Time Attack," Australian Journal of Telecommunications and the Digital Economy, vol. 5, pp. 61-74, 2017. (PUBLISHED)

3. L. E. Jim and M. A. Gregory "AIS Based Danger Theory Framework to Detect Selfish Nodes". (SUBMITTED)

4. L. E. Jim and M. A. Gregory "A Grudger Based AIS Approach to Coerce Selfish Node Cooperation in MANET". (SUBMITTED)

Conference

5. L. E. Jim and M. A. Gregory, "State analysis of Mobile Ad Hoc Network nodes," in Telecommunication Networks and Applications Conference (ITNAC), 2015 International, 2015, pp. 314-319. (PUBLISHED)

6. L. E. Jim and M. A. Gregory, "Packet Storage Time attack-a novel routing attack in Mobile Ad hoc Networks," in Telecommunication Networks and Applications Conference (ITNAC), 2016 27th International, 2016. (PUBLISHED)

7. L. E. Jim and M. A. Gregory, "Modelling of Pain in an Artificial Immune System based MANET," in Telecommunication Networks and Applications Conference (ITNAC), 2017 27th International, 2017. (ACCEPTED FOR PUBLICATION)

1.7 Thesis Composition

The research carried out included a literature review, identification of the research steps, algorithm development, simulations, analysis of the simulation results, comparison and discussion of the results with alternatives found in the literature and identification of future work. The thesis chapters presenting the work carried out are summarized to provide a guide to the thesis composition.

Chapter 1 presents an introduction to the research and includes research aims, objectives and publications. The introduction presents the motivating factors behind the research, why it is important and adds to the body of knowledge and briefly outlines the approach taken.

Chapter 2 includes a literature review that provides essential material about MANET and the current state-of-the-art challenges to assist with understanding the taxonomy of AIS schemes. The literature review is aided by the classification of the AIS algorithms and the work done in MANET. In addition, a few of the pioneering works related to the research into AIS and MANET are presented. Security attacks in MANET are briefly analysed. A brief overview highlighting the trade-off between selfishness and altruism is presented. The analogy between the MANET and HIS is also presented.

Chapter 3 provides the design and development of the AISBA framework and key steps are identified in the chapter sections. Sections 3.4.1 and 3.4.2 briefly describes the nodes in a four-quadrant model that was used for the research and presents the Hop count relation with the Probability of Communication.

Chapter 4 includes the steps taken to detail the AISBA framework, using two types of Trust.

The signals are categorized namely the Safe Signal 1, Safe Signal 2, Danger Signal and Pathogen Associated Molecular Pattern (PAMP) signal. The PAMP signal is utilized to identify selfish nodes. These selfish nodes are identified by the high priority PAMP signal. The trust model is applied in terms of Trust based on interaction as well as Trust generated due to the signals for which nodes are modelled as DCs. Lastly, the statistical analysis in terms of the false positives and false negatives is given in this chapter.

Chapter 5 presents PST, a novel routing attack in MANET, where the attacker node modifies its storage time and thereby does not forward packets to the intended recipient nodes. This attack has been evaluated against various metrics.

Chapter 6 presents the GrAIS Model as this research delves into the need for non-isolating selfish nodes, but instead utilize them for the benefit of the MANET. The GrAIS model takes its groundwork from the AISBA framework presented in Chapter 4 as well as the grudger model of birds.

Chapter 7 deals with conceptualizing pain as a novel concept in MANET, where the nodes in MANET are analogous to the cells in the HIS and thereby prone to pain. A node is said to be in pain when the node's trust and energy are compromised.

Chapter 8 concludes the thesis by providing a summary and a statement of potential future work.

Chapter 2 LITERATURE REVIEW

2.1 Overview

The literature review provides background knowledge that underpins the research carried out, highlights the techniques, methods and approaches that are the state-of-the-art in the research topic area. A brief discussion of different AIS schemes from the current state-of-the-art is provided to highlight the directions being explored. To give the readers an overview of AIS algorithms, Negative Selection, Clonal Selection (CS), DC and Danger Theory are discussed. This chapter is organized into six sections. MANET and the analogy between the MANET and HIS is described in Section 2. An overview of MANET and AIS is provided in Section 3. The concept of AIS is detailed in Section 4, along with insight into MANET and Danger Theory. Section 5 provides details on selected AIS algorithms and their applications. The motivation to use AIS and HIS in MANET is detailed in Section 6. Section7 describes the research carried out into the use of AIS based techniques in MANET technologies. The conclusion and summary are provided in Section 8 and 9 respectively.

Special attention was given to modelling the nodes for reasons made apparent in the state of the art analysis of node modelling provided in Chapter 3. Key concepts in AIS include how MANET protocols are utilized, especially utilizing AIS algorithms, are explained in detail along with their pros and cons. Finally, the different state of the art techniques are briefly described to illustrate how this technique is included in the AISBA framework. Through this translation of individual techniques an efficient and reliable the AISBA framework is achieved.

2.2 MANET Background

MANET is an aggregation of mobile, decentralized, and self-organized nodes. Securing

MANET is a challenge when every node forming the network is a potential menace that could compromise communications using a plurality of attacks.

A MANET can become a self-organized mobile network without the need for fixed infrastructure other than when a connection is needed to a carrier network [13]. Each node in the network acts as both host and data transmitting router after a route discovery process between the source and destination. The routing protocols that have been proposed so far, including Ad Hoc on Demand Vector (AODV), Dynamic Source Routing (DSR), concentrate on the route discovery establishment. MANET security is a concern due to the nature of the distributed traffic management and the requirement that the trust be established with nodes joining the rapidly changing topology.

There are potential weaknesses found in MANETs, and one potential weakness is the susceptibility to a wormhole attack [14, 15] where attackers bypass normal traffic routes and tunnel packets to another area of network. The wormhole route may suffer a lower hop count than normal traffic routes and the attackers manipulate the MANET route priority to perform eavesdropping, denial of service (DOS) and so on. Figure. 2-1 shows the traffic paths for normal traffic and wormhole traffic.

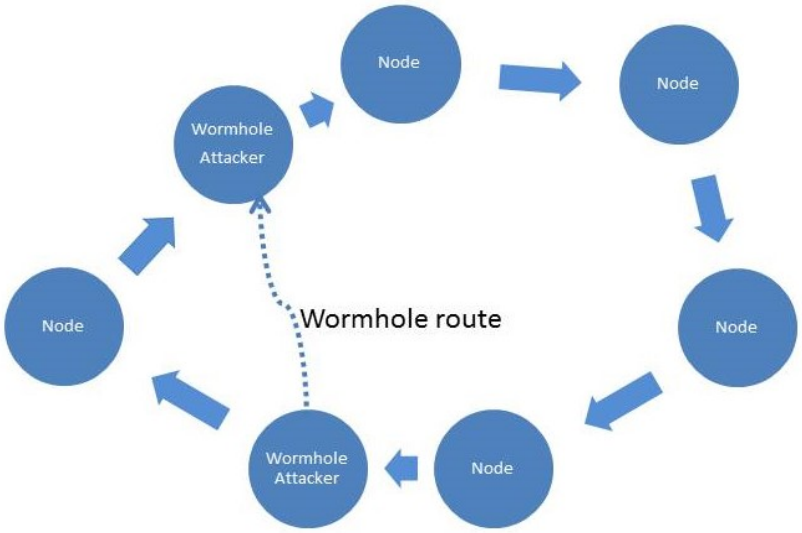

Figure 2-1 Node hops for normal and wormhole traffic

MANET routing protocols are classified as either reactive or proactive according to the route discovery technique used. On-demand protocols are reactive. The presence of a packet available for transmission and the need for a route leads to the commencement of a route discovery process. Table-driven protocols are proactive and routes are computed beforehand and stored in a table so that there will be a route available whenever a packet is to be transmitted. Routing protocols can be broadly classified into stability based, Quality of Service (QoS) based and hop-count based. Hop-count based protocols try to optimize the length of the route which is why a majority of the routing protocols proposed are based on the hop-count metric [16]. The importance of wireless communication systems for reliable mobile communications has increased significantly over the past decade. The dynamic topology offered by MANET provides flexibility and manoeuvrability.

The DSR [2] protocol uses source routing rather than the hop-by-hop routing used by the majority of MANET routing protocols, which eliminates the need for frequent route

advertisement and neighbour detection packets. Applying the concepts and principles of HIS to the development of an AIS based algorithm provides an alternative approach to improve network security in MANET.

2.3 MANET Security Attacks

In this section, an analogy between HIS and MANET is provided and a brief description of the key MANET attacks [4] are given. MANET attacks can be classified into active and passive attacks as seen in Figure. 2-2. Passive attacks involve snooping on the data exchanged in the network without the intention to alter the traffic. This attack is difficult to detect because the network operation is not affected. Passive attacks, gather information about the network, traffic passing over the network or pry on the communication pattern between two or more nodes. A passive attack may lead to an active attack. In a passive attack, the confidentiality of the network could be compromised.

In an active attack, the attacker alters some aspect of the network or the data being exchanged in the network thereby disrupting the normal functioning of the network. The malicious behaviour involves packet modification, injection or destruction.

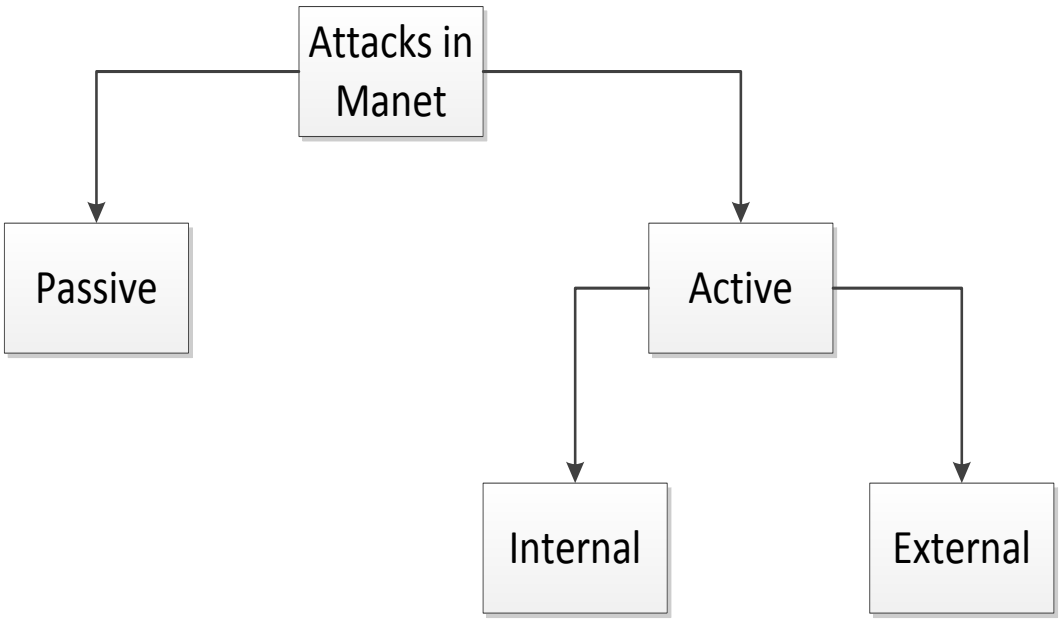

Figure 2-2 MANET security attacks

A brief description of the different types of attacks that occur in MANET is provided in the following sections.

2.3.1 Replay attack

In a MANET, the topology changes permitting replay attacks where the attacker uses the strategy of storing control messages previously sent by a node [5, 6]. The attacker node resends the stored control messages which lead to genuine nodes updating their routing tables with stale information. This disturbs the normal operation of the MANET.

2.3.2 Blackhole attack

In a black hole attack the attacking node sends a false routing message claiming that it has the most conducive route to the destination whereby leading all the genuine nodes to forward their packets to the rogue [7,8].

2.3.3 Flooding attack

In a flooding attack the attacking node sends multiple RREQ messages to a destination node that does not exist in the network [9, 10]. As the destination node does not exist, none of the nodes will be able to send a Route Reply, leading to congestion and a network DOS attack.

2.3.4 Wormhole attack

Wormhole is a type of MANET attack [11] where the attacking node creates a connection to remote nodes as if the two nodes are directly connected to each other. The two distant nodes send false advertising messages to indicate they have a one hop symmetric link between each other. This false information will propagate to other nodes across the network thus undermining the shortest path routing calculations.

2.4 Analogy between MANET and HIS

The analogy between MANET and the HIS is as follows [12]:

- MANET can be considered a metaphor for the tissue of an organism; packets as signals; cells as nodes
- Cells have limited processing, memory, and communication capacity – MANET nodes are similarly resource constrained
- Biological Tissue comprises many cells – high-density MANET implies a substantial number of client devices
- Each cellular phone is prone to failure - cells in biological tissue subject to pathogenic attacks - MANET nodes are prone to failure

- Cells move and reorganize – MANET nodes move and rearrange
- Communication between cells is through the diffusion of signalling proteins and matching of antigenic patterns which is analogous to MANET's where communication is through packet dissemination and matching

2.5 Related approaches in MANET utilizing Hop Count

2.5.1 Multipath Hop-Count Analysis

Multipath Hop-Count Analysis (MHA) is a multipath routing protocol proposed by Kuo et al. that avoids wormhole attacks by using a hop-count analysis scheme as pictured in Figure. 2-3 [18]. MHA is designed to use split multipath routes, thereby causing the transmitted data to be split into separate routes. In MHA, a random variable X is set which represents hop-count values in the Route Reply (RREP) packets. A sample space $U=\{x_1,...x_i,..x_j..\}$ is defined and has a cumulative distribution function $FX(x)$ where α and β represent the lower and upper bound of the cumulative distribution function and s and t represent hop count boundaries.

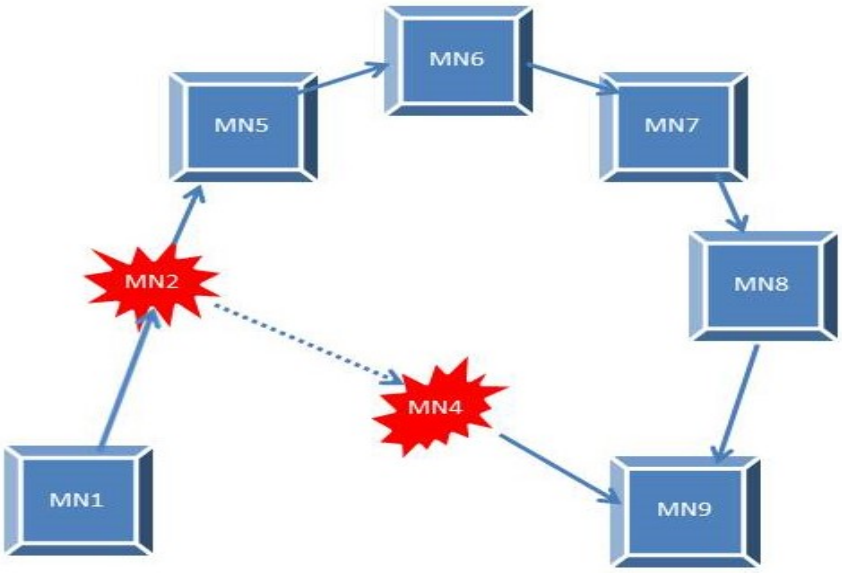

Figure 2-3 Colluding nodes in Wormhole Attack

$$x_i, x_j \in U; \qquad i, j, s, t \in N \tag{2-1}$$

The route paths from RREP packets with hop-count x_h satisfying $x_i \leq x_h \leq xj$ where $h \in N$, $x_h \in U$ are taken as legal paths in the MHA protocol.

2.5.2 Past Interaction Social Analysis

Hsu and Helmy [19, 20] observed that in a network, nodes do not encounter more than 50 per cent of the overall population of the total nodes in a network. As a result, there is less chance of a node encountering other nodes and nodes need to assess the probability that they will encounter the destination node. The authors did an analysis based on network traces of different university campus wireless networks. Their analysis showed that to build a connected relationship graph node encounters are essential.

Haahr et al. [21] proposed social network metrics based on a social analysis of a node's past

interactions. Three components were locally evaluated: a node's "betweenness" centrality, a node's social "similarity" to the destination node and a node's tie strength relationship with the destination node. The centrality of a node in a network is a standard of how well connected it is to other nodes in the network. "Betweenness" centrality calculates the magnitude to which a node lies on the geodesic paths of other nodes.

2.5.3 Probability to Deliver

In the study carried out by Burgess et al. [22] messages are transmitted to nodes in the order of probability for delivery; this is based on contact information. All messages will be transmitted if the connection lasts long enough which in turn results in Epidemic Routing.

Lindgren et al. [17] proposed PRoPHET Routing, which is also probability based, using past encounters to predict the probability of meeting a node again. Frequently encountered nodes have an increased probability and older contacts become obsolete over time. The transitive nature of encounters is utilized where nodes exchange, encounter probabilities and the probability of indirectly encountering the destination is also evaluated.

The cost of the routing path is calculated by defining probability based on node encounters in the study carried out by Khelil et al. and Tan et al. [23, 24].

Grossglauser and Vetterli [25] use "time passed since last encounter" to route packages to destinations. The packet is forwarded to the node that encountered the destination more recently than the source and other neighbouring nodes.

Ghosh et al. [26] propose a hub system, exploiting the fact that nodes tend to move between a

small set of locations identified as hubs. A set of hubs based on each node's movement profile is available to each node on the network in the form of a probabilistic orbit. This probabilistic orbit defines the probability with which a given node will visit a hub. Messages bound for a node are routed toward one of these hubs.

2.5.4 Node location

A location based routing scheme by Lebrun et al. [27] utilizes the trajectories of nodes to predict their future distance to the destination. This technique is also used to pass messages to nodes moving in the direction of the destination

A virtual coordinate system by Leguay et al. [28] uses node coordinates which in turn contain a set of probabilities, each representing the probability of encountering a node in a specific position. The best available path is computed based on this data.

2.5.5 Hop-count in Wormhole routes

Sethi et al. [29] used a hop-count metric to analyse wormhole attacks. See Figure. 2-4 where the advertised path from node *a* to node *d* passes through the nodes *a, b, c* and *d* giving a hop count of three whereby the actual path from node *a* to node *d* passes through nodes *a, b, e, f, g, h* and *d* making the actual path of length six hops. This difference in hop-count between the advertised path and actual path can be useful for the detection of wormholes.

Choi et al. [30] suggested a scenario where nodes will keep track of the conduct of its neighbours. RREQ is sent by a node to the destination using its neighbour list. However, if the RREP is not received back within a stipulated time, the presence of a wormhole is identified and the route is added to the source node's wormhole list. Each node maintains a

table which consists of a RREQ sequence number and neighbour node ID. After sending RREQ, the source node sets the Wormhole Prevention Timer (WPT) and waits until it overhears retransmission by the neighbour node. The maximum amount of time for a packet to travel one-hop distance is *WPT/2*.

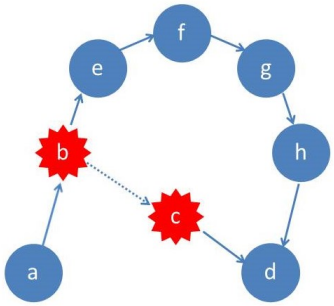

Figure 2-4 Hop-count metric for wormhole

2.5.6 Routing protocols using hop mechanism

Destination Sequenced Distance Vector (DSDV) [34] is a proactive routing protocol where all routes to destination nodes, hop-count, and next-hop to the destination are maintained in a table. The table is updated by the periodic exchange of messages between neighbouring nodes. As paths are readily available to all destinations in network there is less delay associated with the path set-up process. Another disadvantage is due to the need to regularly update routing tables the node battery life is shortened.

Optimized Link State Routing (OLSR) [33] is a proactive routing protocol. The neighbors and link are detected by Hello messages that are broadcast to discover single-hop neighbors. Message flooding can be limited to nodes within a certain distance (in terms of number of hops). Nodes can also examine the header of a message to get information pertaining to distance (in terms of number of hops) of the message source.

DSR [31] is a reactive or on-demand routing protocol that discovers the routes only when it has packets to be forwarded to the destination. The RREQ packets contain data regarding the intermediate nodes and the hop-count. This protocol also optionally uses a flow ID option that allows packets to be forwarded on a hop-by-hop basis.

AODV [32] is a reactive routing protocol where routes are established on demand. To send a message to a destination, RREQ messages are broadcast to intermediate nodes. The intermediate nodes in turn rebroadcast the messages to their neighbours. When a neighbouring node receives a request and it already has a route to the desired destination, it sends back a message to the source node through a temporary route thereby creating routes from various neighbouring nodes. The source node then selects the route that has the lowest hop-count.

2.6 Tradeoff between Selfishness and Altruism in MANET

Routing protocols developed for MANET can be classified as proactive, reactive and hybrid. The effect of node selfishness on routing and node resource utilization efficiency has not been studied adequately. In [35] misbehaviour in MANET was first identified and defined and the focus of this work was to alleviate node misbehaviour. The research found in the literature appears to focus on how to detect and isolate selfish nodes. These methods do not penalize the selfish nodes nor to coerce the selfish nodes to forward packets. The malicious nodes are rewarded if they're identified and removed from routing paths. In [36] a review of node selfishness in MANET is provided. This research summarizes existing approaches to dealing with the selfishness problem and the authors provide a proposed solution to mitigate the selfishness problem. The operation of DSR [37] is explored and as energy depletes node

selfishness occurs. Various types of selfishness are defined and the problems arising because of selfish nodes co-existing in the network is investigated.

In [38] the data flow between the MANET nodes is observed and when a selfish node does not forward a packet, the neighbour node waits for a pre-defined threshold number of packet transmission failures to be exceeded before triggering an alarm.

In [39] the impact of selfish nodes on the MANET QoS is explored. This work analyses parameters including throughput, average hop count and packets dropped. The hop count increases as the selfish node concentration increases. The authors establish that there is an increment in the number of packets dropped along with a substantial reduction in throughput as the selfish node concentration increases.

In [40] the MANET nodes are encouraged to be altruistic and the nodes are given positive or negative scores depending on their behaviour. The altruistic nodes utilize their energy to relay for other nodes, but they relay for selfish nodes only once. This approach does not call for the participation of selfish nodes for any communication.

In [113] the author employs a theoretical account that considers how birds clean each other of parasites in hard to reach spots, thus helping with individual and group survival. The author defines three different model behaviours:

1) Sucker - Birds that blindly help other birds without expecting anything in return.

2) Cheat - Birds that take advantage of all the help they can get but do not offer anything in getting even.

3) Grudger - Birds that help others and recall who they have served. In case the same bird does not reciprocate, they will not help that bird again.

2.7 Artificial Immune Systems

"AIS are intelligent and adaptive systems inspired by the immune system toward real-world problem solving. AIS are adaptive systems inspired by theoretical immunology and observed immune functions, principles and models, which are applied to complex problem domains [41]."

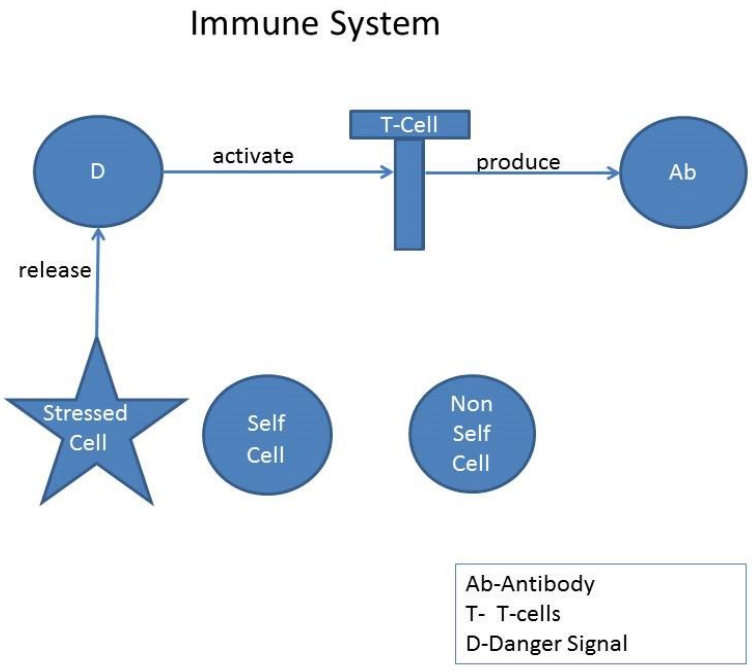

Figure 2-5 Danger Theory model

A recent immunological discovery called Danger Theory now paves the way for more efficient, second generation AIS. The Dendritic Cell Algorithm (DCA) [42] is a biologically

inspired technique, developed to detect intruders in computer networks. The DCA is based on a metaphor of naturally occurring DCs, a type of cell which is native to the innate arm of the immune system [41]. DCs are responsible for the initial detection of intruders, including bacteria and parasites by responding to the harm done by the invading entity. Natural DCs receive sensory input in the form of molecules, which can indicate if the tissue is healthy, or in distress. These cells have the ability to combine the various signals from the tissue and to produce their own output signals. The output of DCs instructs the immune system responder cells to deal with the source of the potential damage. DCs are excellent candidate cells for abstraction to network security as they are the body's own intrusion detection agents.

To improve the performance of AIS algorithms a “danger project” has been commenced based mainly on the immunology Danger Theory, which states that the response type of the immune system to the incoming pathogens occurs due to the existence of danger or safe signals from the body tissues affected by the pathogen, as illustrated in Figure 2-5 [41].

DCA is a danger project contribution that utilizes the DC role in HIS as forensic navigators and important anomaly detectors. DCs are defined as antigens presenting lymphocytes in the innate immunity; these lymphocytes play a key role in either stimulating or suppressing the adaptive immunity T-cells and hence controlling the immune system’s response.

The field of AIS has achieved importance as a branch of Computational Intelligence since its inception in the 1990s. The four major AIS algorithms on which research is centred are: (1) Negative Selection Algorithm (NSA); (2) Artificial Immune Networks (AIN); (3) CS Algorithm; and (4) Danger Theory and DCA. AI brings together the disciplines of

Immunology, Computer Science and Engineering. Over the past decade research into the Immune System has gained popularity as a vehicle for novel solutions to complex issues. The highly distributed, adaptive nature of the immune system includes capabilities such as learning, memory and pattern recognition, which are solid foundations for an artificial equivalent. AIS outcomes require both integration of immunology and engineering to transform the complex evolved mechanisms found in the HIS.

Forrest et al. [43] proposed a negative selection method to distinguish self from non-self, based on the generation of T-cells. This approach was applied to the problem of virus detection in a computer and raised the profile of negative selection approaches. Following the work by Forrest et al. variations of the NSA have been developed with the essential properties of the original NSA remaining.

AINs are another popular AIS approach that was based on the work by Farmer et al. into an immune network model [44] and an early immune network algorithm was designed by Ishida [45]. Timmis et al. [46] redefined the artificial immune network. Castro et al. [47] proposed the Clonal Selection Algorithm (CLONALG) which is based on a CS principle processes like those found in Genetic Algorithms including clone, mutation and reselection.

The AIS community has produced a multifaceted set of immune inspired algorithms to solve computational as well as real world problems. Castro and Timmis [48] offered a detailed analysis of the Immune System and a presentation of current AIS algorithms. Tarakanov et al. [49] provided an insight into the mathematical basis of immunocomputing. Ishida [50] reviewed immune network models and highlighted the benefits of each approach.

2.8 AIS Algorithms

Select AIS algorithms are described in this section.

2.8.1 Negative Selection

In the biotic or biological immune system T-cells are initially formed in the bone marrow and on maturation they move to the thymus. The phase of T-cell evolution is characterized by expressions provided by T-cell receptors. Whenever the Pre-T-cells and thymus cells interact this land thymus cells interact this leads to Pre-T-cell multiplication and divergence. Then these T-cells undergo negative selection to eliminate T-cells that are activated by self in the thymus. Although variations of negative selection have been proposed, the process described in [51, 52] remains in usage.

Gao et al. [53] proposed enlarging non-overlapping detectors to obtain non-self coverage. Let the detector centre be O_j, then detector j will have a maximum radius of r_j. The detectors with large radii have higher fitness. Ma et al. [54] describe a mechanism to produce useful detectors that are randomly produced and the unmatched antigen is placed into a detector space called the feedback detector. The feedback detector will be eliminated in case it matches self-strings. Once the feedback detector becomes mature it will be utilized to match antigens. When the feedback detector acquires a match on further antigens, it becomes a legitimate detector. Simple Evolutionary NSA (ENSA) and basic ENSA [54] are NSA variations and the functionality of Simple ENSA is to generate detectors capable of identifying corrupt data. When a detector seeks to match data it can lead to wayward or abnormal changes in the detector and this detector will be discarded. The evolution of the

next generation of detectors takes place through mutation, positive selection and negative selection. Such evolutionary inception loops to generate detectors until a wayward change is noticed. In Basic ENSA, in addition to the next generation detector set a randomly generated detector is also added. By including the additional detector searches can take place in the global space as well. ENSA finds its use in hardware/software segregation in embedded systems.

Caldas et al. [55] proposed a variant of the NSA where a repository database is used to store perceptible performance indexes for an enterprise. There will be a set of cells known as decision cells, which will be responsible for extracting decisions from the repository database and provide feedback about the decision to the repository database. Each decision issue is represented by a decision cell which in turn is composed of x decision receptors. The approach proposed consists of two stages: learning and operation. In the learning stage, the decision maker selects the decision cells based on the information in the repository database. The cells form the initial reservoir of self-cells, that is a decision cache to be stored in the repository database for later usage. In the operation stage, the decision creator requests decision cells from the repository database and present decision related problems to the decision cells for resolution.

Graaff et al. [56] proposed the Genetic Artificial Immune System (GAIS). Here the counterpart of lymphocyte is known as an artificial lymphocyte. The four states in which an artificial lymphocyte exists are: immature (no priority), mature (medium priority), memory (high priority) and annihilated (low priority). The bit string of an artificial lymphocyte is randomly generated and is made to undergo either positive selection or negative selection.

Based on the Hamming distance of the nearest self-pattern to the artificial lymphocyte, it will be assigned a distance threshold value. Whenever a match happens with a non-self-pattern the Hit counter of the artificial lymphocyte is incremented to find its matching ratio.

GAIS also uses Genetic Algorithms (GA) to evolve artificial lymphocytes. Each artificial lymphocyte is related to a chromosome and the randomly generated artificial lymphocytes will constitute an existing GA population.

Amaral et al. [57] uses GA to generate a detector in a real valued NSA. Every possible detector set is linked to a chromosome. Each gene is a pointer to a y dimensional detector set. The radius for each detector set is computed by using a decoding function and Monte Carlo integration [58, 59] is used to calculate the volume of the detector set.

2.8.2 Artificial Immune Networks

Jerne [60] suggested that the immune system can attain immunological memory due to the presence of B-cells. These B-cells prompt each other as well as restrain connected cells to control over production of B-cells. This is required to keep a stable memory.

Hunt and Cooke [61] suggested a scheme comprising a bone marrow object, a network of B-cell objects and antigen population. Bone marrow objects randomly initialize the B-cell population. The antigen population that is present in the system will be randomly picked and introduced to a spot in the B-cell network. Cloning of B-cell objects occur if they can bind to the antigen population.

Pacheo et al. [62] designed an Abstract Immune System Algorithm. There are four strategies

necessary for the effectiveness of this model: (1) the affinity between the epitope of an antibody or prototype of an antibody; (2) the restraining of an antibody during epitope recognition; (3) the affinity between antigen and antibody; and (4) the nature of cells to die in the absence of communication. A given antibody type will be prompted or deleted by referring to the recruitment threshold or death threshold.

Omni-aiNet [63] is applied to solve singular and multi-objective problems. The advantages identified were: (1) a new grid mechanism to control the spread of a solution in the objective space; (2) adjusting the size of the search space based on a predefined suppression threshold; and (3) axiomatically adapting the investigation of the search space.

Taking advantage of the multi-population property of aiNet, the Multi-objective Multi population Artificial Immune Network (MOM-aiNet) for bi clustering was designed [64]. The advantage of MOM-aiNet is that several sets of non-subjugated solutions are returned in contradiction to a single set of non-subjugated solutions. The subjugation is used to compare the quality of solutions for a given issue, thereby enabling it to measure the solution set given by MOM-aiNet. Out of the data set one row and one column is randomly chosen so that MOM-aiNet produces y subpopulation of one bi cluster. In the algorithm for each subpopulation y clones subject to the mutation process will be developed. Three steps are involved in mutation which would be randomly chosen with equal probability: (1) delete a row of the column; (2) incorporate a row; and (3) incorporate a column. Whenever the number of non-subjugated elements becomes greater than y clones a distance based restraining process occurs so that a small and locally diverse sub-population is maintained.

Stibor et al. explored the compression quality of aiNet [65]. Using the Parzen window estimation and Kullback-Leibler divergence a similarity measure between the data set (input) and aiNet dataset (output) was introduced. A Parzen window estimator helps find the probability densities of the input and output datasets.

2.8.3 Clonal Selection Algorithms

According to the CS Theory when the original lymphocyte is activated by binding to the antigen, clonal expansion of the original lymphocyte occurs. During the development of the lymphocyte, if any clone with antigen receptors corresponds to the molecules of the organism's own body, it will be eliminated. With the clonal expansion of B-cells the average likeness increased for the antigen that sparked the clonal expansion through likeness maturation. Thus, the B-cells more effectively respond to antigens. Somatic hyper-mutation and the Selective mechanism lead to likeness maturation. Somatic hyper-mutation leads to a miscellany of antibodies by introducing random changes to the genes. Only those genes with a higher accord for the encountered antigen will survive. CLONALG was initially introduced in [66] and described in [47,67].

Ciccazzo et al. [68] suggested a variant of CLONALG termed Elitist Immune Programming (EIP). EIP is an extension of immune programming and the concept of elitism is borrowed from the immune inspired algorithm and is introduced to EIP. A new category of hyper-mutation operators and network based coding is used in EIP. Any hyper-mutation operator can only act on one node or link at a time. This work leads to the proposition of ten ad hoc network based hyper-mutation operators: add-parallel, add-series, delete component, mutate-

component-value, copy-component-value, add-random-component, mutate-component-kind, link-modify, shrink and expand-node. The EIP algorithm was applied to a synthesis of topology and size of analog electrical circuits. Based on the experiments the circuits designed by EIP were an improvement over that achieved using Genetic Algorithms.

Halavati et al. [69] included symbiosis to CLONALG and uses specified antibodies, which are an approximate solution only, as they may not contain the data required. Each antibody will have just one property. Later the algorithm randomly selects an antibody to be included in an assembly. By using repetitive steps an assembly with the required properties is built, however, in instances where the algorithm is unable to build an assembly, antibodies with random values are created for the missing component parts and a new assembly is created. The technique of utilizing partially specified antibodies stems from the deduction that a problem can be broken into smaller problems and solutions to these smaller problems may provide an improved overall solution to the overarching problem.

The approach in [70] proposed a variation of CLONALG for software mutation and testing that utilizes the notion of “memory individuals” that steer to the identification of an antigen rather than utilizing the notion of the CLONALG memory individuals per antigen. An antibody population is initialized with p tests either randomly generated or pre-specified. A periodic check is done by the algorithm searching for antibodies that will kill at least one mutant program. A Mutation Store is used to assess the freshness of an antibody and antibodies with a higher similarity score are added to the memory set to be returned to the tester. The productiveness of this method was compared against an elitist GA and the results showed that the proposed methodology produces a higher mutation score with lower

computational cost.

The Trend Evaluation Algorithm (TEA) proposed by Wilson [71] is similar to CLONALG however; it incorporates a long-term memory pool as well as short term memory pool by multiplying all of the bound trackers. The processes of Apoptosis and Mutation in the TEA occur across all population members. Consider the case where an antigen Ag containing 40 fictional price movements and ten Trends (T1-T10) is built to test the ability of the TEA to identify price trends. Antigen Ag is divided into four subsets Ag1, Ag2, Ag3, and Ag4. Ag1 contains two simple trends T1 and T2 and the more complex trends are involved in Ag2, Ag3 and Ag4. Experiments were done to test the algorithm's capability to discern price trends as well as to probe the algorithms influence over the long-term memory pool.

2.8.4 Danger Theory based Algorithms

Danger theory is another self/non-self theory that differs from other theories in how the system should respond. The salient characteristic of Danger Theory stems from the principle that the immune system does not respond to non-self but does respond to danger. This theory evolves out of the consideration that there is no need to assail everything foreign. In this theory,danger is measured by the distress signals sent by cells in the event of damage or unnatural death.

Matzinger, proposed Danger Theory in 2002 [72] and highlighted that the "foreignness" of a microbe is not the main factor that ignites a response and "selfness" is no assurance of tolerance. The fundamental idea in Danger Theory states that antigen presenting cells are triggered by danger/alarm signals from sore cells. Danger signals will not be sent by healthy

cells or by cells experiencing normal cell death.

The Two-Signal Model extended by Bretscher et al. [73] explains the Danger Theory in a different way where two signals are needed to activate the lymphocytes: (1) antigen recognition; and (2) co-stimulation. Signal 2 indicates that the antigen is threatening.

The Danger Theory has its own disadvantages and Aickelin et al. [74] proposed applications of the Danger Theory that highlight:

- The presence of an Antigen Presenting Cell (APC) is required to present a danger signal.
- A danger signal does not have to be dangerous.
- Danger signals can be positive or negative (presence or absence of signal).
- An estimate of nearness may be used to imitate the danger zone.

Conceptual ideas were also proposed on how the Danger Theory can be used for anomaly detection. Founded on the Danger Theory, an immune response is always triggered by danger signals. Low or high memory use, fraudulent disk activity and so forth could indicate danger signals. The Immune System can react to the antigens in the danger zone once a danger signal is produced. After the dangerous components are identified, they are then sent to a special part of the system for further verification. Another application of the Danger Theory used in intrusion detection can be found in [75].

Danger Theory has been applied to data mining problems [76]. Consider the case where a

user is browsing a set of documents where each document has a set of attributes. When AIS is implemented the antibodies in the system are used to detect the attributes. Each document browsed by the user will be dispensed to the antibodies. When the user expresses interest in the present document a danger signal is raised and antibodies matching the antigen (attribute in the present document) are triggered and become active. Wearisome document attributes will endure the auto reactive antibodies. Finally, AIS learns to become a good filter when searching for documents.

Prieto et al. [77] used a goalkeeper strategy in the Danger Theory Algorithm (DTAL) that takes into account danger signals, lymphocytes and the danger zone. This technique was used in robot soccer, when the ball is on the source side (tissue) an alarm signal (Signal 1) will be triggered. When the ball (antigen) is taken by the opponent to the penalty side (danger zone) Signal 2 will be triggered. When both signals are received the lymphocyte is actuated to clear the ball. This strategy showed a performance above 90%.

The work in [78] highlighted an application of the Danger Theory to accentuate the effectiveness of an e-mail classifier system. In web-mining the use of various types of media may cause various signals to be released, but in an e-mail system an abnormal email may release a "fascinating" signal of one category. The strong pertinence of these features constitutes a form of the Danger Theory.

2.8.5 Dendritic Cell Algorithms

The main role of DCs as antigen presenting cells were identified by Steinman and Cohn [79] where DCs are comprised of leukocytes which are present in all tissues. They are endowed

with a disparate hematopoietic lineage and function in various tissues. Inside various tissues, DCs segregate and mature when triggered appropriately; later they relocate to secondary lymphoid tissues where they present antigen to T-cells to induce an immune response.

The immature DCs occupy body surfaces and are commonly present in an immature state and are unable to stimulate T-cells. Once the foreign pathogens are processed and obtained by the immature DCs as seen in Figure 2-6, they migrate to the thymus and the spleen where the immature DCs mature and stimulate an immune response. As explained in [80, 81] inflection between the various states of DCs is enabled by the recognition of signals, including PAMP, danger signals, apoptotic signals (safe signals) and inflammatory cytokines. These signals are explained as (1) PAMPs activate the immune response, thereby protecting the host from infection; (2) danger signals are released during tissue cell damage, their strength is lower than PAMPs; (3) safe signals are given out when programmed/normal cell death occurs; and (4) inflammatory cytokines are given out when general tissue distress occurs and amplify the effect of the other three signals. The immune response of the T-cell is determined by the corresponding weights of the four signal types. Semi-mature DCs have a suppressive effect while mature DCs have an accentuating effect.

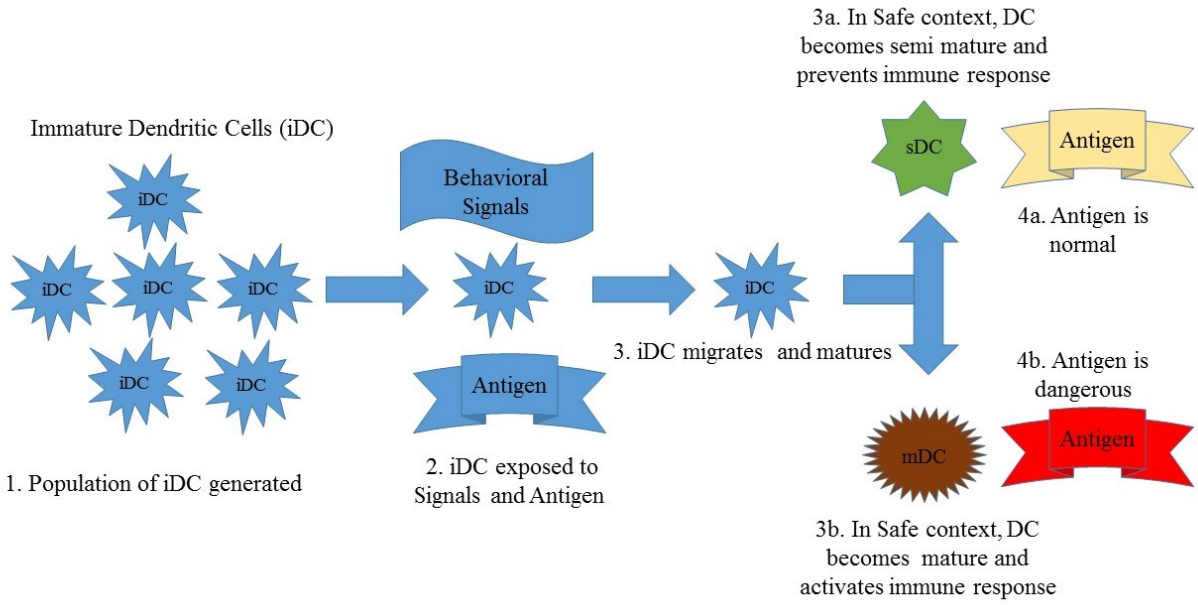

Figure 2-6 Dendritic Cell Algorithm Schematic

The first DCA was presented by Greensmith et al. [82] and it involved combining various signals to investigate the current circumstance of the environment and non-parallel sampling of another data stream (antigen). A fuzzy margin occurs in accordance with the concentration of co-stimulatory molecules as an indicator for a DC to stop antigen collection and migrate to a virtual lymph node. The DCA works on the input signals with presumed weights to produce output signals. A value of 1 is assigned if the cumulative mature signal is greater than the cumulative semi-mature signal and vice versa. The mature context presentation of that antigen is calculated relative to the total number of antigens.

The DC is designed as a Libitissue tissue server [83]. There are three stages in this algorithm: initialization, update and aggregation. Initialization deals with setting initial values and the update stage is sub-divided into tissue update and the cell cycle. The Libtissue tissue server comprises the tissue update and cell cycle. Data from the source is given to the tissue server

through the tissue client. The appearance of new data in the system leads to the provision of input signals for the population of DCs. The cell cycle is a distinct process that occurs at a user defined rate. When the antigen data is processed the cell cycle and tissue update process stops. In the final stage aggregation of the collected antigens occurs together with analysis and the Mature Context Antigen Value (MCAV) per antigen is derived.

Gu et al. [84] used DCA on the KDD 99 [85] data set after two additional functions were added to the system for optimization: antigen multiplier and a moving time window. The antigen multiplier makes several copies of the antigen, to overcome the problem of "antigen deficiency" that can be given to DCs. On each iteration, new signals are calculated using the moving time window. Based on the results the antigen multiplier and moving time window have equal effect on the DCA using the KD 99 data set.

Oates et al. [86] devised a DCA approach for a robot classification problem. Robotic DCA is designed as a stand-alone physiological module for compatibility with comprisal design. The Advanced Robot Interface for Applications (Aria) library's [87] "wander" design is extended with two extra modules: image processing and DCA execution. MCAV coefficients are output by the DCA module approximately once per second. PAMP, safe and danger signals are used as input to the DCA. PAMP originates from the image processing module and the safe signal originates from the Laser Range Finder (LRF). A sonar array having a 360 degree field of view (FOV) is the source of the danger signal and the antigen is an integer number which can be uniquely identified by the segment of the test pen. The DCA approach used helped the robot to steer away from obstacles in its path.

The authentic DCA is highly speculative and the Deterministic Dendritic Cell Algorithm (dDCA) [82] attempts to overcome this by using two sets of input signals as well as antigens. The DC is subjected to identical input signals. Here an array is used in order to store the antigen value and count of times the DCs have collected the antigen. There are three parameters in the dDCA-weight scheme for processing signals, outputting DC values and the number of DCs.

The work in [88] depicts the affinity of DCA towards the architecture and operational requirements of sensor networks. Based on this variation, ubiquitous DCA (UDCA) was proposed to detect attacks on sensor networks and its features include:

Signals from multiple data sources are collected by DC. New output cytokines are accumulated at the maturation stage of each DC. The linking of antigens with context information is done by UDCA. The extent of node misbehaviour is detected by UDCA via signals generated.

2.9 MANET and HIS

Fundamental aspects of an ad hoc wireless network include its lack of fixed infrastructure, design and challenges including security [89] and the lack of a centralized control mechanism adds to the complexity of fault and security intrusion detection and correction. The dynamically changing nature of mobile nodes causes an unpredictable topology that requires frequent route changes, network partitioning and protection from increased packet loss. The security attacks on MANET networks utilize opportunities provided by the wireless mobile infrastructure in which nodes can join and leave at will using dynamic requests [90]. Energy

efficient routing algorithms can be tricked into routing through compromised nodes if the node indicates high power when the other battery powered nodes are showing varying power levels [91]. The failure of one node may affect the entire MANET and this adds to the network design complexity, especially as the probability of network partitioning increases as node power levels fluctuate. Mobile node power supply limitations and energy depletion is a major factor affecting the lifetime of the ad hoc network [92].

HIS has been identified as a source of models, functions, and concepts that inspire AIS algorithms which can be used to secure both host-based and network-based systems [1]. However, it is not only important to utilize the HIS when creating AIS-based algorithms as much as it is important to produce high performance algorithms [41]. Therefore, creating a balance between utilizing HIS and introducing AIS-based intrusion detection algorithms are a crucial issue that would be valuable to investigate because MANET properties raise security issues to a level above those associated with fixed networks. The AIS properties such as being self-healing, self-defensive and self-organizing provide an opportunity to meet the challenges of securing MANET [93].

2.9.1 Introduction and Need for AIS Conceptualization in MANET

An ad hoc network is formed by a group of nodes that do not require any predefined infrastructure to maintain network connectivity. One of the many advantages of MANET is the absence of dedicated fixed nodes to support packet forwarding and routing. MANET nodes act as both host and router. The application of MANETs span from military operations to the commercial sector such as rescue/emergency missions. Ad hoc networks can also link

with a temporary multimedia network to share information amongst users in a conference or classroom. Short-range MANET has simplified the connection between mobile devices, thereby replacing the need for cumbersome cables.

This autonomous nature of MANET makes it vulnerable to malicious attacks, thereby making MANET susceptible to active as well as passive attacks. In a MANET, every node must be prepared for an encounter with an adversary either directly or indirectly. The mobility characteristics of the ad hoc network enable the nodes to roam independently, thereby making the task of identifying and tracking compromised nodes difficult. Security attacks on MANET nodes can result in compromised nodes that function incorrectly and potentially generate traffic with false routing information.

The AIS is derived from the natural HIS and is a branch of Artificial Intelligence. The research carried out in the application of AIS helps to bridge the gap between engineering, science and immunology. Immune system characteristics provide an attractive research focus for applications in engineering and science. The evolution of AIS research has its roots in the study carried out by Farmer, Packard and Perelson [44].

2.9.2 Developments in AIS Based MANET

The DCA ability to act as an anomaly detector algorithm inspires further investigation of the biological model to introduce improved DC inspired algorithms [1], which could detect other types of security attacks [6] in a MANET. In addition, many of the MANET characteristics and properties are similar to the innate immunity abstract features; such as the openness and susceptibility of each to different types of danger attacks [5].

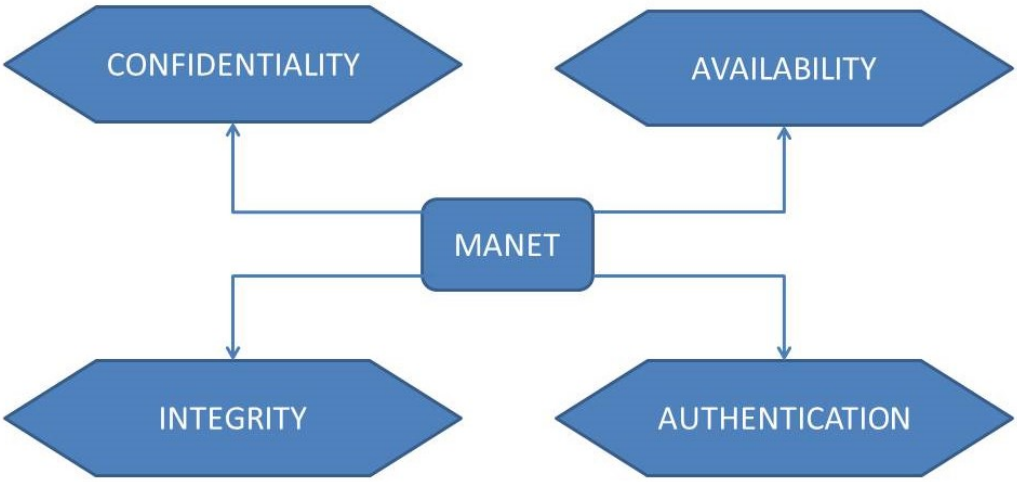

Figure 2-7 MANET security goals

MANETs share the same basic security goals that occur in other network types. The need for confidentiality, authenticity, integrity, availability, non-repudiation and access control as illustrated in Figure 2-7, which is the same as in other network types [94] and is generally determined by the importance and sensitivity of applications used or data transmitted. Network control, management, and security goals are harder to achieve in a MANET than in conventional networks [95] due to the mobile decentralized nature of the network.

Sarafijanovic et al. [96] investigated the use of AIS to detect node misbehaviour in MANET using the DSR and the AIS algorithms with negative selection and CS. In this proposed system as illustrated in Figure 2-8, each DSR node implements an instance of the detection system, and runs it in two stages. In an initial stage, the detection system learns about the normal behaviour of the nodes with respect to the DSR protocol. During this stage, the node is supposed to be in a protected environment in which all nodes behave properly. From the packets received or overheard, the node observes the behaviour of its neighbours and creates

positive antigens. Towards the end of this learning stage, the node runs the negative selection process and creates its antibodies, known as Detectors.

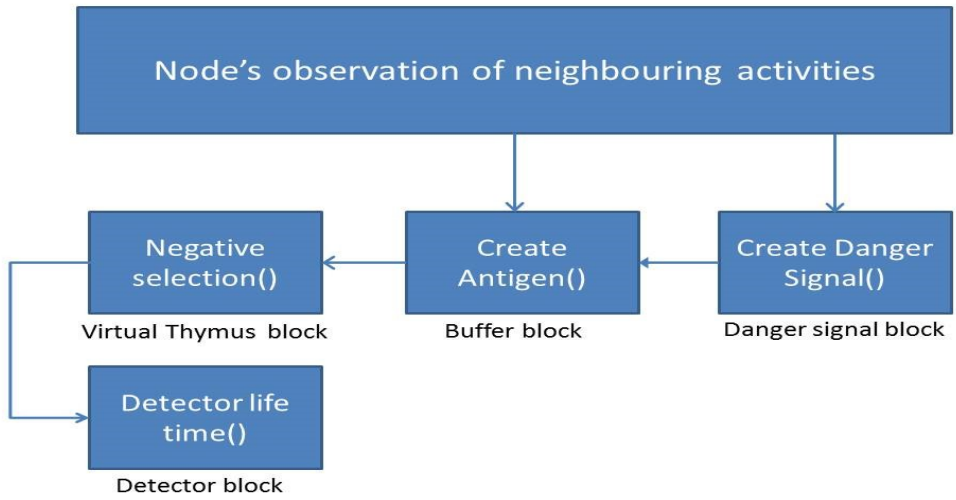

Figure 2-8 Detection system

After the initial stage, the node may leave the protected environment and enter the second stage where detection and classification are carried out. In this stage, the node may be exposed to misbehaving nodes. The Detectors created in the learning stage are used to check if newly collected antigens represent the behaviour of good or bad nodes. In case an antigen, created for any neighbour during some time interval, is detected by any of the Detectors, the neighbour is identified as doubtful in that time interval. If there are too many doubtful intervals for a neighbour, that neighbour is classified as misbehaving. This triggers the CS process in the node that made the classification. In this process, the node adapts its detectors to improve misbehaviour detection.

The work carried out by Hoffmeyer et al. [97] uses the self-non-self model negative selection process and some form of danger signal. In the system proposed, the Transmission Control

Protocol (TCP) connections play the role of self and non-self cells. TCP is a computer networking protocol that provides reliable data packet exchange between two networked devices that communicate over a multi-hop network. One connection is represented by a triplet encoding the sender's destination address, the receiver's destination address, and the receiver's port number. A Detector is a bit sequence of the same length as the triplet and matches a triplet if both have contiguous equal bits. Candidate Detectors are generated randomly; in a learning phase, Detectors that match the correct (i.e., Self) triplets are eliminated and this is done offline, by presenting only valid TCP connections. The Detectors that are not eliminated have a finite lifetime and die unless they match a non self triplet, as in the IS. The danger signal is also used and it is sent by humans as confirmation in case of potential detection. This is a drawback, since human intervention is required to eliminate false positives, but it allows the system to learn about changes in the self.

In the implementation of IDS [98] to secure MANETs the authors present an approach based on the paradigm of HIS. This is achieved by using a Mobile Agent which they identify as the Immune Agent (IA). The IA consists of four processes based on the scenarios encountered in the wireless ad hoc domain.

- Detection process

This is triggered when a connection between two nodes is established.

- Classification process

The next security process is the classification of self or non-self.

- Blocking/Isolating Process

The aim of this process is to block and isolate a node which is classified as malicious based on the standards stored in the IA.

- Recovery Process

The IA takes a snapshot of the data recovery file when it successfully attaches to the new node that intends to join the wireless domain. When a change in the node's system is detected a classification for the pattern that caused the change is also determined. This approach uses memory, where data has been fed into a database and the same data is fetched and used in the recovery process. The following profiles are created for the purpose of a security approach based on immune inspired properties:

- Gene Profile. This profile would contain the recurring events needed to establish a connection system. This is similar to self-cells in the Immune system.

- Detector Profile. This profile is used to recognize non-self which is similar to HIS T-cells.

- Non-Self Profile. This profile would contain events that harm the system.

The Immune agents capture the self and non-self patterns during the monitoring and capturing phase and we learn that $U = S_f \cup N_f$ represents the collection of patterns monitored while packets transfer, it contains both self and non-self patterns. $N_f = \{n_{f1}, n_{f2} \ldots n_{fm}\}$, $S_f = \{s_{f1}, s_{f2} \ldots s_{fn}\}$ represents the set of all self and non-self patterns captured by the Immune Agent. To simulate the T-cells the Immune agent will be equipped with detectors that are

randomly generated. $Đ = \{d_1, d_2 \ldots d_m\}$ represents the set of the generated Detectors, $Đ' = \{d'_1, d'_2 \ldots d'_m\}$ the set of matured Detectors.

The NSA is used to collect the matured Detectors to ensure that the generated Detectors do not match any self. The next step is CS where a Detector will be cloned if it attains a score after matching to non-self. The algorithm is as given below:

- Let: d'_i score = 0
- For $N_f = \{n_{f1}, n_{f2} \ldots n_{fm}\}$; bind d'_i to n_{fj}, (for i, j=1, 2…, m);
- If d'_i detects n_{fj}, then d'_i score++; end if
- While $\{d'_i \text{ score} \geq \text{max score}\}$ do clone d'_i // proliferation phase
- $d'_i = d''_i$;
- If d''_i match s_{fi} ;($1 \leq i \geq n$); then delete d''_i ;// negative selection
- Else $Đ' = Đ' + d''_i$ // Update the Detectors Profile

To utilize the Danger Theory concept, the immune agent keeps a replica of the data necessary to regain a node. Consider β, a system with components at time t: $\beta_t = \{\beta_1, \beta_2 .. \beta_n\}$. A copy of β_t is available in the Immune Agent Database. Therefore, any change in the system components can be identified. Let ε be a change that occurs in the system after time Δt. The Immune Agent checks the system and observes $\beta_t + \Delta t = \beta \pm \varepsilon$. As ε is not recognized, it is a suspect pattern and will be included in the non-self-set to be blocked in future. This approach is not implemented and simulated so the accuracy of this approach cannot be validated.

Nauman et al. [99] proposed using a DC approach in combination with a BEE algorithm. The scouts and foragers of the BEE algorithm are used in the DC formation. This algorithm uses a dynamic detector set and the DCs are modelled to sample the antigens (scouts) from the body tissues (node). During this phase, both self and non-self-antigens are sampled. At startup random Detectors are generated which are in turn subjected to negative selection with regard to self-antigens represented by the semi mature DCs.

Using negative selection to generate Detectors involves computational overhead and generating Detectors in a dynamically changing environment like MANET is not viable.

According to Ye et al. [100] two IAs, the detection agent and counterattack agent are entrusted with detection as well as reaction. The detecting agent may be viewed as a T-cell lymphocyte while the counterattack agent may be viewed as an antibody. Whenever the detection agent finds an invader, instructions are sent to the counterattack agents. The behavioural patterns of nodes identified are as follows:

- Node Q received message P recorded as Recv (Q, P)
- Node *Q* sends message *P* recorded as Send (*Q*, *P*)
- Node *Q* keeps message *P* recorded as Keep (*Q*, *P*)
- Node Q modified message *P* recorded as Modify (*Q*, *P*)
- Node *Q* deletes message *P* recorded as Delete (*Q*, *P*)
- Node *Q* generates new message *P* recorded as Make (*Q*, *P*)

- Node Q verifies message P recorded as Verify (Q, P)
- Node Q stores message P recorded as Store (Q, P)
- Node Q broadcasts message P recorded as Broadcast (Q, P)

Message R is the reply of the message P recorded as Reply (P, R). The behaviour patterns of attack nodes are kept in the Immune Memory Library to represent different attack methods:

- Method 1: Recv (Q, P), Delete (Q, P). Node Q receives the message P and deletes it without transmitting it. This is an Interrupt Attack.
- Method 2: Recv (Q, P), Modify (Q, P), Send (Q, P). Node Q upon receiving the message P modifies it and then transmits it. This is an Error Message Attack.
- Method 3: Recv (Q, P), Reply (P, R), Send (Q, P). Node Q receives message P and sends the message via the wrong route. This is called a Black Hole Attack.
- Method 4: Recv (Q, P), Keep (Q, P), Send (Q, P). Node Q receives message P and then transmits it after keeping the message for some time. This can ensue in a Hidden Attack.
- Method 5: Make (Q, P), Broadcast (Q, P). Node Q makes and broadcasts a large number of messages in a short time, which leads to node overload. This is called a Denial of Service Attack.
- Method 6: Store (Q, P), Modify (Q, P), Send (Q, P). Node Q modifies the details of the route and transmits again which will result in other nodes receiving error filled

routing messages.

The detection agent records the behaviour of each of the neighbouring nodes. When the node behaviours do not match, they are analysed using the Immune Strategy Library.

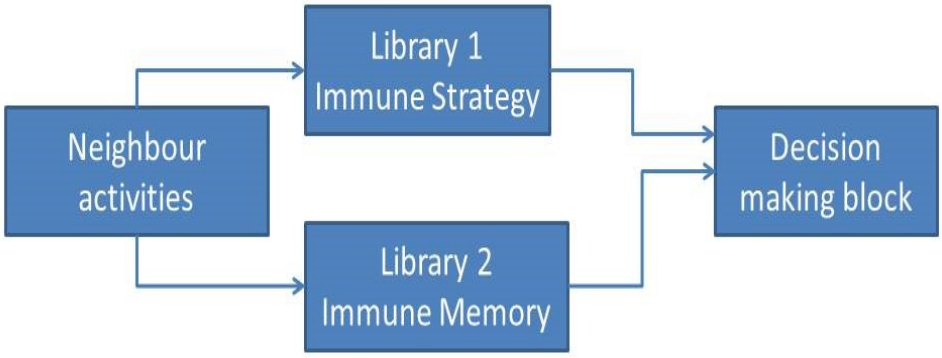

Figure 2-9 Immune libraries

The creation of the Immune Memory Library and Immune Strategy Library as illustrated in Figure 2-9 is not mentioned in detail in [100] and the Detection Agent cannot record the behaviour of a particular node as there are other nodes in addition to the neighbouring nodes which could be compromised. If the Detection Agent was to record each node's behaviour this would result in a considerable computational overhead. This has not been simulated so the accuracy and feasibility of the approach is left to future work.

Fatemeh [101] suggested a combination of AIS and GAs that are used to adjust to alterations in network topology and Spherical Detectors are generated to handle non-self-space. The technique employed to generate Spherical Detectors is an area for future research that might be employed to identify the equivalent to protein compound antigens that exist in body cells alongside pathogens.

The innate immune system uses built in knowledge to combat against infections and a danger

signal means damage caused by self-cells due to antigens coming from non-self. In Danger Theory, the recognition of pathogens is not enough to get a response from the adaptive immune system, but an additional sense of danger is needed before the body reacts to any infection caused by pathogens.

Nauman [102] proposes two approaches based on AIS called BeeAIS and BeeAIS-DC. BeeAIS utilizes negative selection to detect anomalies in MANETs and with the use of negative selection, the profile of the system behaviour during normal routing is found.

Antigens extracted from incoming traffic in the network are created from the packet header data. Here the antigens are modelled as one of three different types; scout antigen used to detect anomalies relating to scouts and forager antigens of two types used to detect changes to the source path. Antibodies and Detectors are created by combining four gene values as random numbers. Matching functions are when the interaction between antigen and antibody is measured in terms of distance in Hamming shape space.

The two stages of BeeAIS operation are the Learning Phase and the Protection phase. In the Learning Phase the system behaviour is identified under normal routing conditions where each node monitors traffic in order to gather the information needed to make self-antigens. When a scout is received, a node may form a scout antigen and when a forager is received; forager antigens are made. A node could receive the same self-antigen many times. Hence it matches the newly formed antigen with the antigens that have been previously collected from the traffic flow.

After the end of the Learning Phase a set of Detectors are generated using a negative

selection process with the self-antigen set collected during the Learning Phase. The Detectors will be generated randomly and only those that do not match with self-antigens are kept.

In the Protection Phase the nodes collect antigens from the incoming traffic and carry out measurement of their affinity with the Detector sets. Whenever a match occurs, it indicates an anomaly is present. However, this approach fails as the algorithm learns the system behaviour only once and as a result, during the Protection Phase newly observed behaviour is declared as malicious by the system.

Whenever a node receives a forward or backward scout it creates an antigen. After extracting the relevant fields from the scout header an antigen is created. The fields which are extracted include the scout source, destination, length of route and node ID of the previous hop. DCs are formed when a node sees a scout and the DCs are initialized with the following attributes:

- DC Antigen: The sampled antigen from the scout is attached to the DC.
- DC Life: The DCs are assigned a short lifetime and they die a natural death after that.
- DC State: Upon instantiation, a DC is an immature DC and when antigens are sampled and when safe signals are present, the DC transitions to a semi-mature state. During the exposure to danger signals DCs transform to mature DCs.

As the system starts, a set of random Detectors is generated by the node and the Detectors undergo a negative selection process during which antigens are identified. The Detectors that match with antigens are eliminated and the resulting Detectors match with non-self-antigens. Mature DCs are used to activate T-cells. During matching the T-cell detector is transformed

to become an activated Detector.

The approach proposed in [102] doesn't describe the role of an activated Detector sufficiently and the process carried out after the Detector becomes active is not adequately explained. Negative selection requires a Learning Phase which is not practical in a dynamic MANET. How the role of the Detector to curb malicious activities in the MANET is to occur is not adequately explained.

Ansari et al. in [103] use the concepts of CS and danger signal for misbehaviour detection using DSR. The protocol events of a node are mapped to HIS elements. The genes of a node are designed based on the performance of the network, the node's observations of neighbouring nodes. These genes form the ground to detect if a node is misbehaving. Antigens are represented by a pattern of observed events generated by the protocol.

The events generated at the monitored node when it receives a packet originating at sender node are as given below

- i=RREQ sent
- j=RREP sent
- k=RERR sent
- l=DATA sent
- m=RREQ received
- n=RREP received

- o=RERR received
- p=DATA received

The antigen set is represented as:

- D={mmimmmimmjmimmplplp,plplplplplplplplp}
- G_1=Num (m)
- G_2=Num (m*(i+j))
- G_3=Num (p)
- G_4=Num (p*l)

Where G_k denotes the k_{th} gene, *Num* denotes the number of occurrences, * denotes "zero" or more occurrence. Each bit in the antigen set *D* is termed as a nucleotide. Antibodies are generated randomly after which they are passed for negative selection.

Ab1= {1010100011, 0001101100, 1100100010, 0110001010}

Ab_2= {1010111000, 0101100100, 1010101010, 0110101110}

$Self_{Ag}$= ({0000100000, 0000000100, 0000000010}, {0000000000, 0000000000, 0000100000, 0000010000})

$Self_{Ag}$ denotes self-antigen and the node saves this information when it is in the learning phase. Whenever a node experiences packet loss a danger signal is generated. The criteria to realize the self-antigen is not mentioned. The question arises if an antigen is generated by an

attacker node in the same time-frame. This would result in all nodes considering SelfAg to be a trustworthy pattern and generate antibodies that cannot accurately detect misbehaving nodes. Whenever a '1' occurs in an antibody pattern which matches with the '1" in Self antigen, the antibody will be deleted.

Sarafijanovic et al. [104] attempt to detect node misbehaviour by making the nodes learn what normal behaviour is in a protected environment. In this scenario, a self-antigen pattern would be generated and antibody patterns are deleted if there is '1' in every position the antigen has a '1'. Here again the question arises if the same antigen pattern could be generated by an attacker node.

In the approach proposed by Kim et al. [105] each node extracts a set of feature vectors y out of normal network traffic. Each feature vector is represented by a hyper sphere with a fixed radius in the feature space. At each time slot $\Delta t_i \; \varepsilon \; t$ every node extracts a q dimensional feature vector $y_{i.}$ and (2-2) describes the network state.

$$y_i = (y_i^1, y_i^2, \ldots, y_i^q) \tag{2-2}$$

Where $y_i^k \; \varepsilon \; [0, 1]$ is a measurable feature vector. The feature space is represented by $S_p \subseteq [0.0, 1.0]^p$ where $y_i \; \epsilon \; S_p$ is associated with an antigen.

A feature vector $y_i \; \varepsilon \; S_p$ at time t is termed normal if it belongs to a normal network state. To generate a set of negative Detectors $N(t)$ every negative Detector $n_j \; \epsilon \; N(t)$ is defined as a hypersphere (a_j, b_j) where a_j is the centre of the hypersphere and b_j is the radius.

Let $P(t)$ be the set of positive antigens. The Niche NABC algorithm [106] takes $P(t)$ as input

and generates a set *N(t)* of mature negative detectors. Immature food sources are created so that there will be a minimum overlap with positive antigens. When the quality of food sources cannot be improved further the food source will be abandoned. In this approach, there is an offline learning phase and an online learning phase. The offline learning phase is run in a protected environment and leads to the creation of negative Detectors. This approach does not map a food source to any of the routing or MANET parameters.

Anass et al. [107] proposes a detection generation algorithm. In each generation, the DCs deliver a set of elements that are of fixed size randomly chosen from the antigens. Based on the context of the element which is presented a number of operations are established to allow memory Detectors to detect intrusive behaviour. When the context of the element is dangerous then the algorithm checks if the memory detectors can detect the antigen. If the danger element is not detected by the memory Detector the algorithm checks if the mature Detectors are able to detect this element. If there is a mature Detector which can detect the element then this mature element is added to the group of memory Detectors. In case the presented element is harmless the algorithm checks if this element is detectable by the memory Detectors to remove the corresponding detector.

The context upon which the element is classified to be “dangerous” is not detailed. This experiment is not validated hence it is not possible to verify the approach.

Visconti et al. [108] suggests a type 2 fuzzy set based algorithm for detecting misbehaving nodes that is triggered by network danger signals and antigen presenting cells. The approach in [109] is used in order to capture the real behaviour of a node and the experts provide the

Footprint of Uncertainty (FOU). A red region indicates misbehaviour of the network pattern, a yellow region indicates suspicious behaviour and a white region indicates normal behaviour. The binding process invokes the helper T-cells to measure the actual changes of the network parameter and find the region (Red, Yellow, and White) to which the Interval type 2 fuzzy parameter is closer. Therefore, to conclude if a node is good or bad I2FM is built for the whole network based on all M network parameters. The proposed approach is a work in progress.

In [110] an immune system approach has been proposed for securing MANET. The Immune Agent consists of three profiles: gene profile, non-self-profile and Detector profile. The gene profile consists of the frequently occurring events for connection establishment. The Detector profile is similar to T-cells in the human body that detect the non-self. The non-self profile contains events that harm the system. The Immune Agent captures and stores the information pertaining to the protocol during the Training phase (Secure) as well as in the insecure phase. The Immune agent will be equipped with Detectors that are randomly generated. The NSA ensures that the generated Detectors do not match self. The Detectors that come out of the negative selection stage are cloned whenever they attain a score detecting non-self.

The Combined Immune Theories Algorithm (CITA) [111] utilizes the basic principles of well-known immune theories including DCA, CS, and NSA. This algorithm is compared with the Secure Ad hoc on Demand Distance Vector algorithm (SAODV) [114] and improved performance is demonstrated. DCA is used to obtain context information. DCs are associated with a subset of neighbouring nodes called elements, which are responsible for DC maturation. Element subsets are monitored using adjacent Immature Detectors (ID), adjacent

Mature Detectors (MTD) and Memory Detectors (MMD). The network is first configured with trusted nodes during the learning phase. Each node will have a set of detectors. CITA utilizes several parameters that are initialized during the learning phase, including the number of detectors available and the definition of alarm signals.

2.10 Interaction between pain, nervous system, and immune system

Pain can be conceptualized as the alarm system for the body. When there is a threat of harm, the brain interprets that threat and signals the alarm. This alarm stimulates an avalanche of responses like a stress-response. One portion of the stress-response sets the activation mode of the nervous system into flight or fight. In its most basic configuration, the nervous system prepares the body to move away from or to fight a threat. The duration and magnitude of the stress have a great influence on whether or not the immune system will be affected by this avalanche, either suppressed or enhanced. Only once the alarm has been tripped does the connection between the nervous system and immune system [112] come into play.

This link between the nervous system and the immune system is strong. The “fight or escape” system is focused on one over-riding instinct: to survive. When tailed by a lion or burned by a stove, the reactions and responses are short-term and immediate. To divert energy to muscles for running or moving, the body can shut down more long-term, high-energy processes like digestion and immunity functions. This kinship is more complex than a simple shut down operation because during the initial stress-response the nervous system fine tunes and enhances functions of immunity. Only as the stressor continues, the nervous system triggers the shutdown of immunity and begins to dismantle it.

In MANET, which is analogous to HIS, pain can be modeled using factors like trust and energy. These are two crucial factors that are required for the proper functioning of any routing protocol, which benefits the network as a whole. The energy of a node participating in the routing protocol is critical. Only if a client has sufficient energy can it duly forward/deliver packets. A packet cannot be simply forwarded to a node just because a client has sufficient energy. This is where the trust factor becomes important. A node should be deemed trustworthy in order to receive packets from the source/intermediate node. This scenario has been conceptualized in MANET through the research presented in the following chapters.

In some events, pain may enhance immunity or feed into an over-active immune system. The interaction between pain, the nervous system and immune system is complex.

2.11 Summary

Techniques based on Immunity are becoming more popular and emerging as a new branch of Artificial Intelligence. The NSA is being continuously applied and modified to solve problems. This review highlighted that NSAs are utilized for new detector generation schemes and a broad discussion is required between the biologists, scientists and engineers to learn fresh ways of applying AIS. This chapter gives a critique of the various AIS approaches applied to MANET and discusses how research is tackling the difficult issues surrounding MANET security. Most of the approaches identified are either work in progress or have not been validated demonstrating there is scope for further study.

The literature review provided in Chapter 2 has established the background principles, algorithms and direction being taken to further develop AIS as an efficient approach to

MANET security. This review discusses the representative state-of-the-art AIS schemes.

The literature review has also provided a discussion along the relevant routing and MANET protocols based on their relevance to the research project. A brief introduction of different AIS principles, their classification and a concise discussion of the algorithms with common active periods has been shown.

The chapter has also included a discussion on the different AIS techniques and its applications in MANET as well as other fields.

RMIT UNIVERSITY

Chapter 3 ANALYSIS OF MANET NODE STATE

3.1 Overview

This chapter provides research carried out to answer Research Question 1 on the design, modelling and analysis of the proposed AIS framework in MANET. Here MANET is analysed in terms of quadrants and the nodes are dispersed amongst the quadrants. In Section 3.2, the motivation for the node analysis is detailed. Section 3.3 presents and discusses approaches that have utilized the transmission hop count. Section 3.4, takes advantage of the idea of a Markov model and transforms the hop count metric into the probability of communication or probability of node nearness. The relation between the hop count and the probability of communication is illustrated. Section 3.5 concludes the chapter, highlighting the importance of the hop count metric.

3.2 Analysis of MANET nodes

Consider a MANET consisting of several quadrants as seen in Figure 3-1, where the nodes are dispersed within the quadrants. There can be numerous points N spread along the x axis where N is a real number. A point C on the number line from $–x$ to x is considered. Assuming there is a node at point C, and the node moves along the x axis and when the node is at point C the position of the node can be represented as C_X. A node can also move along the positive plane of the x axis and can be at positions such as C_{X+1}, C_{X+2}, etc. The node can also move along the negative plane of x axis and can be at positions such as C_{-X}, C_{-X-1}, C_{-X-2}, etc. At each hop the node can move along the positive or negative plane $(+x/-x)$ of the respective axis it is present in. This movement of the node follows a Markov chain model known as the drunkard model. Similarly, the node can move along $+y/-y$.

The movement along $+x/-x$ can also be considered as moving along

$$\text{Node}_{\text{Left(L)}}/\text{Node}_{\text{Right(R)}} \quad (3\text{-}1)$$

Similarly, the movement of the node along *+y/-y* can be considered as moving along

$$\text{Node}_{\text{Up(U)}}/\text{Node}_{\text{Down(D)}} \quad (3\text{-}2)$$

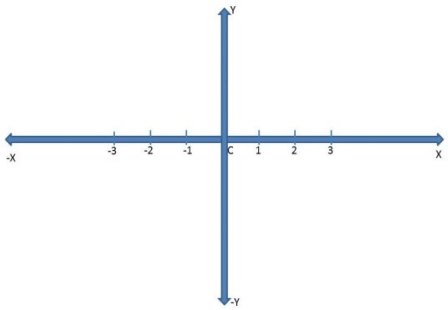

Figure 3-1 Node positions

If the node takes one hop (+1/-1) at a time to reach its immediate neighbour, there is a probability of 0.5 when moving along *+x/-x* and *+y/-y* axes respectively.

3.2.1 Node Movement Probability

The node can either move by +1 hops or by -1 hops. By analysing (3-1) and (3-2) we can find the probability that the node moves by +1/-1 hop as a square matrix of non-negative values. Combining (3-1) and (3-2)provides (3-3) a doubly stochastic matrix which identifies with the probability of node movement (P_{NMOVE}).

$$P_{\text{NMOVE}} = \begin{matrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{matrix} \quad (3\text{-}3)$$

Based on P_{NMOVE} the different states, or the state transition due to node hops, can be identified which gives the state diagram shown in Figure 3-2 for the node in *'L'* state and *'R'* state.

As shown in Figure 3-2 the *L* state is followed by the *R* state and similarly the *U* state is

followed by the *D* state. From this we get a transition matrix which is doubly stochastic.

This change of node state can be identified as a Markov chain whereby the node undergoes transitions from one state to another. Considering a node hopping along C_{X+1}, C_{X+2}, C_{X+3}, etc. such that the future state is only dependent on the present state and independent of the past or previous states, then $P(C_{n+1} = c | C_1 = c_1, C_2 = c_2 ..... C_n = c_n) = P(C_{n+1} = c | C_n = c_n)$ if both conditional probabilities are well defined i.e. if $P(C_1 = c_1, C_2 = c_2, ..., C_n = c_n) > 0$.

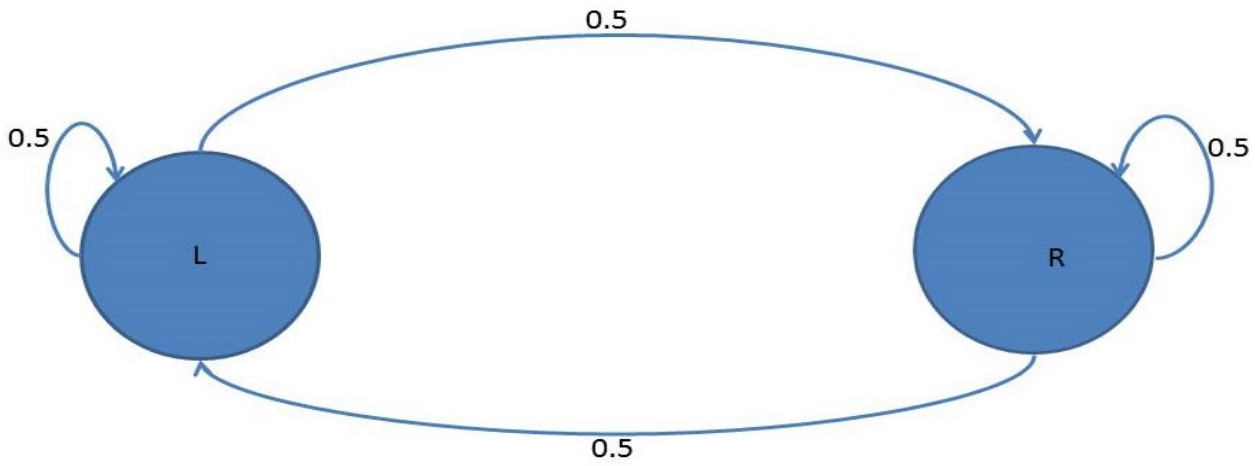

Figure 3-2 State diagram for L and R states

3.2.2 Node State Classification

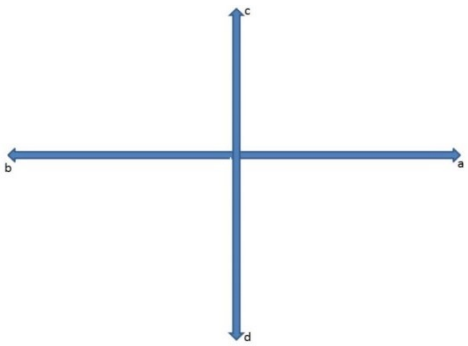

Figure 3-3 Node states

Consider nodes N_a, N_b, N_c and N_d to be at states *a*, *b*, *c*, and *d*. In Figure 3-3 state *b* is

accessible from state *a*, $a \rightarrow b$, if $P_{ab}^{n} > 0$ for every $n \geq 0$. This implies there is a possibility of reaching state *b* from state *a* in *n* hops. If state *b* is not accessible from state *a* then

$$P^{n}_{ab} = 0 \; \forall \; n \geq 0 \tag{3-4}$$

Let us consider *P* (ever visit $b|C_0$=a) i.e. the probability of going from state '*a*' to '*b*' in '*n*' hops then

$$P_{ab}^{n} = \Pr(\mathrm{Xn} = \mathrm{b}|\mathrm{X0} = \mathrm{a}) \tag{3-5}$$

The *n* hop transition probabilities satisfy the Chapman- Kolmogorov equation for any *k* such that $0 \leq k \leq n$ then

$$P_{ab}^{(n)} = \sum_{r \varepsilon S} P_{ar}^{(k)} \, P_{rb}^{(n-k)} \tag{3-6}$$

Where *S* is the state space of the Markov chain and *r* is any intermediary state in between state *a* and state *b*. If *a* is accessible from *b* and *b* is accessible from *a*, we say that *a* and *b* communicate, which is an equivalence relation. The equivalence relations that can be obtained are

$$\mathrm{a} \leftrightarrow \mathrm{b} \tag{3-7}$$
$$\mathrm{a} \leftrightarrow \mathrm{b} \text{ implies } \mathrm{b} \leftrightarrow \mathrm{a} \tag{3-8}$$
$$\mathrm{a} \leftrightarrow \mathrm{b} \text{ and } \mathrm{b} \leftrightarrow \mathrm{c} \text{ together implies } \mathrm{a} \leftrightarrow \mathrm{c} \tag{3-9}$$
$$\mathrm{a} \leftrightarrow \mathrm{b};\ \mathrm{c} \leftrightarrow \mathrm{d} \text{ implies } \mathrm{a} \leftrightarrow \mathrm{d} \tag{3-10}$$

Let's look at (3-7) in more detail; Assume $a \leftrightarrow b$ and $b \leftrightarrow c$ this means there exists $n \geq 0$ so that

$$P_{ab}^{n} > 0 \tag{3-11}$$

And $m \geq 0$ so that $P_{bc}^{m} > 0$. It is now possible to get from state *a* to state *c* in $m+n$ hops by going from state *a* to state *b* in *n* hops and from *b* to *c* in *m* hops

$$P_{ac}^{n+m} \geq P_{ab}^{n} P_{bc}^{m} > 0 \quad (3\text{-}12)$$

$$P^{n+m} = P^{n} P^{m} \text{ and then } P_{ac}^{n+m} = \sum_{l} P_{al}^{n} P_{lc}^{m} \geq P_{ab}^{n} P_{bc}^{m} \quad (3\text{-}13)$$

This accessibility relation divides states into classes and this shows that within each class all states communicate with each other. Let *D(t)* be the distance in meters between a neighbouring node and the node that last transmitted a *RREQ/RREP* message. The positive/negative value indicates the interval $-F(t) \leq D(t) \leq F(t)$. *L(t)* is the maximum distance the node can cover at time *t*. *F(t)* is the number of hops the node can make during *L(t)*. Hop-Count is a measure of distance in the network and is given by

$$F = \frac{D}{L} \quad (3\text{-}14)$$

Based on the hop count, the probability density function can be calculated as follows, consider *X* and X_0 to be node positions and consider the node to be at an initial position X_0

$$1 \Big/ \sqrt{4 \pi D t_F}\, e^{-(X - X_0)^2 / 4 D t_F} \quad (3\text{-}15)$$

(3-15) shows that the probability of finding the node at *x(t)* is Gaussian and as the motion of a node is random, the node can be considered as a Brownian particle.

$$\frac{d(X, t|X_0)}{dt} = D \frac{d^2(X, t|X_0)}{dx^2} \quad (3\text{-}16)$$

The nodes keep changing their states and the probabilities associated with state changes are called Transition probabilities. The probability of going from state *i* to state *j* in a single step is a single step transition. For a single step transition

$$P_{ij} = P_r(X = j|X_0 = i) \quad (3\text{-}17)$$

Consider *N* to be the total number of network nodes

$$K = \sum_{i,j=0}^{n} F^{Pi,j} \quad \text{where } 0 < n \leq N \tag{3-18}$$

K is the node nearness factor. The probability of a node being able to reach its next hop neighbour can be represented as the Probability of node nearness or Probability of communication (P_{com}). (3-16) can be rewritten as

$$P_{com} = \frac{\ln(K)}{\ln(F)} \tag{3-19}$$

As the probability ranges from 0 to 1 ,the ideal value for K after simulation and analysis, it was identified that the value of node nearness factor $K=2$, and when $K=2$ performance improved.

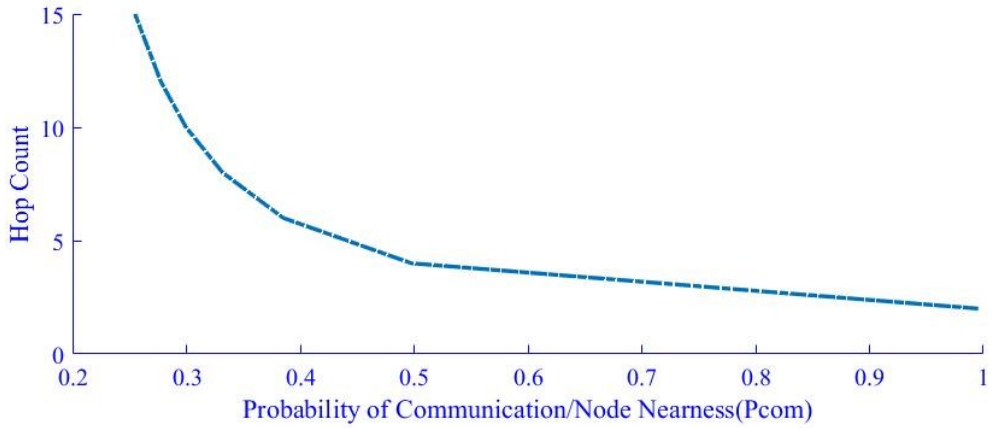

Figure 3-4 Hop-count versus probability of communication

3.2.3 Hop-Count versus Probability of Communication/Node Nearness

The results of a simulation using MATLAB and NS-3 are shown in Figure 3-4 and when the probability of node nearness increases the distance the node travels is reduced thus minimizing the number of hops taken to reach its neighbouring node. As the hop-count increases the Probability of Communication/Node Nearness (*Pcom*) decreases which in turn indicates the hops taken by the node increases.

3.2.4 Delivery Time versus Delivery Cost

An increasing number of hops from source to destination increases node battery consumption due to an increased likelihood of retransmissions and the result is an increase in the delivery cost [115,116]. For different values of *Pcom*, from Figure 3-5, it is shown that if the time to deliver is high the associated delivery cost is also high. For an increasing value of *Pcom* there is a decrease in delivery time as the node can communicate with the neighbouring node faster.

3.2.5 Routing Overhead versus Node Velocity

As shown in Figure 3-6 for different values of *Pcom* as velocity is increased the routing overhead increases. As the value of Probability of node nearness increases the routing overhead reduces as there is a lower chance for a message delivery failure. As the node moves towards its neighbour node for a higher value of *Pcom*, route discovery costs are minimized.

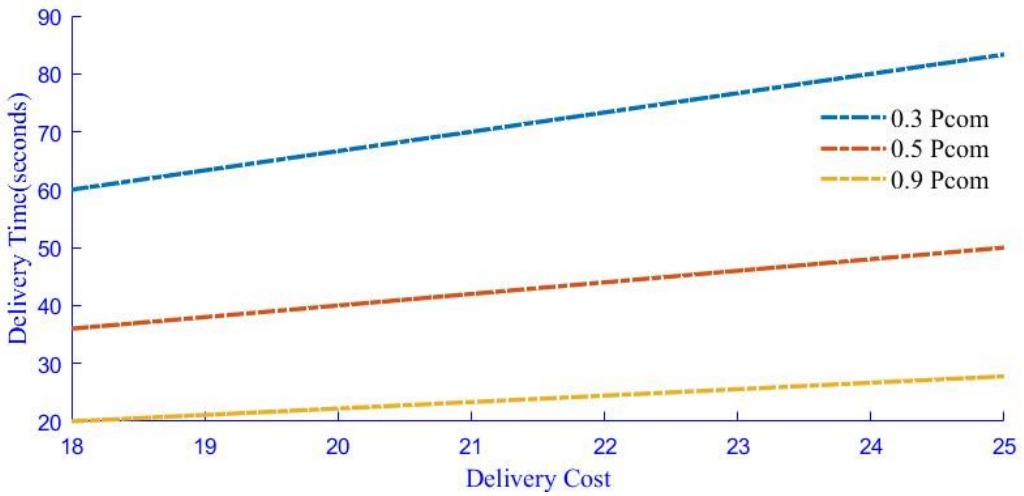

Figure 3-5 Delivery Time versus Delivery Cost

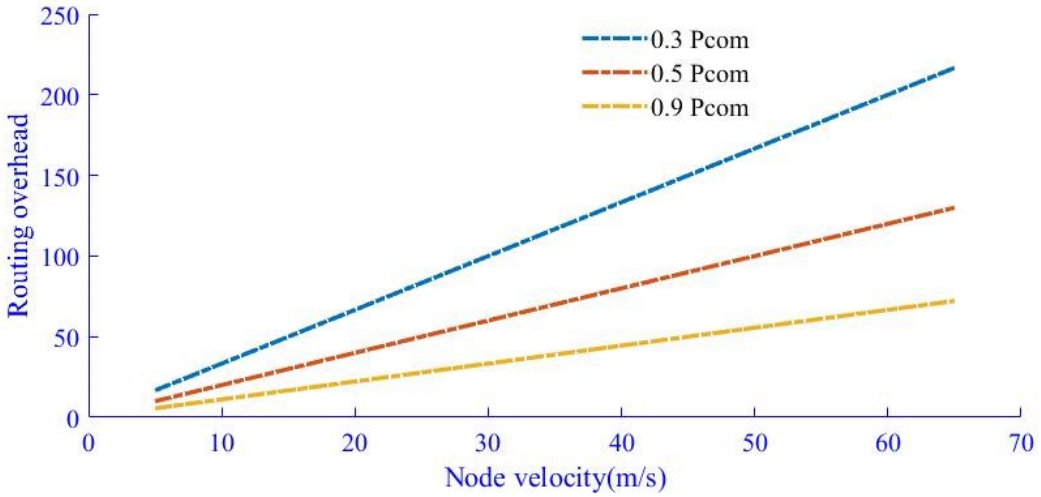

Figure 3-6 Routing overhead versus Node Velocity

3.3 Summary

In this chapter, the node communication process is analysed based on their states and hop-count. The analysis is based on the Markov random walk model and the importance of hop count for efficient communication is analysed to understand the hop count dependency in terms of the probability to reliably communicate to a neighbour node and other associated impacts. An analysis of the results is presented along with a discussion of reliability and scalability.

In this chapter, we have highlighted the importance of hop-count thereby leading to the Probability of node nearness and observed from the analysis that *Pcom* for a node plays a major role in network efficiency. In the following chapters, the hop-count metric has been included in the development of improved security techniques thereby making MANET more robust and secure.

Chapter 4 AIS ENHANCED SECURITY

4.1 Overview

In this chapter, a new approach using AIS is presented which mimics the strategy of the HIS and Trust models to distinguish between a genuine node and selfish or compromised nodes. The research presented responds to Research Question 2 and the proposed framework utilizes the DCA principles. This algorithm utilizes the concept of Probability of Communication described in Chapter 3, for the development of an AISBA and is organized into the following five sections. Section 4.2 details existing MANET attacks. Section 4.3 describes the AIS based detection scheme. Section 4.4 and 4.5 analyses the detection system followed by the conclusion and areas that require future study in Section 4.6.

4.2 Artificial Immune System Based Algorithm

4.2.1 Inspiration

The immune system distinguishes between non–self harmful antigens and non–self harmless antigens using specific mechanisms. The adaptive immune system and not the innate system could be the natural system and able to spot danger signs because it is hard-wired through evolution. In humans, however, due to genetic mutations that take place over time the immune system may be put into a deficit state that could result in catastrophic failure or contribute to an evolutionary update that incorporates the genetic mutations, with both positive and negative outcomes.

In the adaptive immune system, white blood cells mediate the protection forces that might identify as an army of specialized able forces that have been trained to identify enemies and hence mount a defense. The protection forces work with the innate system to cleanse out the

damaged debris and rely on the natural system for alarms in the case of imminent danger as well as re-training if necessary.

The damage signals are generated from the affected tissue, for example, a splinter results in the destruction of many cells due to skin penetration. A splinter may be accompanied by bacteria. The splinter is comparatively a non-self-harmless pathogen, while the bacteria are a non-self substantially harmful pathogen. The signal that spontaneously notified the able specialized forces comes from the dying cells. Cell death leads to the release of proteins that are not supposed to be on the outside surface of a living cell. The released proteins in turn alert the innate and adaptive systems about the necessity for a clean-up and immune response. The nature of how the living cells died is investigated and the able specialized forces are trained to carry out rectification activities. Therefore, the adaptive system will start to learn about the non-self proteins it finds near the damage and in turn, will train more specialized able forces near the abuse. The bacteria that followed the splinter can grow and lead to more damage; they may also have some innate proteins which in turn help to hunt them down. More signals will be generated raising the alarm level so that specialized able cells will hunt more belligerently for threatening pathogens.

In some cases, the bacteria are vanquished rapidly, and all that remains is the splinter. Unlike bacteria, the non-self splinter is not going to cause more damage; the damage has already been done, and no more proteins will be released by damaged cells as they die. Minor irritations may occur, but the remaining splinter may not be a bother causing more cell damage causing the alarm to be raised again. The non-self and non-harmful splinter will still be confronted by the specialized able cells and they, in turn, begin to learn that the proteins

associated with the splinter aren't all that harmful. Thereby the specialized cells start to become allergic to the splinter proteins. This is how the HIS learns the difference between harmful non-self pathogens and non-harmful ones. The reason behind the cell death is the key to understanding if training the able specialized forces is required.

During normal cell death or apoptosis, the proteins are not released, but are bundled and deactivated to avoid raising the alarm while being swept by the garbage collection teams of the human body.

4.3 AISBA Trust Model

Unlike some of the prior work carried out into AIS, where the learning phase is carried out once, there is a need in a dynamic topology like MANET to utilize an approach that permits each phase to be carried out in an iterative and repetitive process. The proposed algorithm follows a reactive approach as having a learning phase makes the algorithm have a large computational overhead. As the MANET topology is dynamic, thereby it is not efficient to learn things beforehand. If a node behaviour is not known before then that node is malicious during that time interval as seen in Section 2.7. The blacklisted node could have a route error. In such scenarios, designing a security algorithm based on the learning phase is inefficient. Therefore, the security algorithm proposed in this paper is a reactive one which accurately verifies genuine cases of route error and identifies a malicious node.

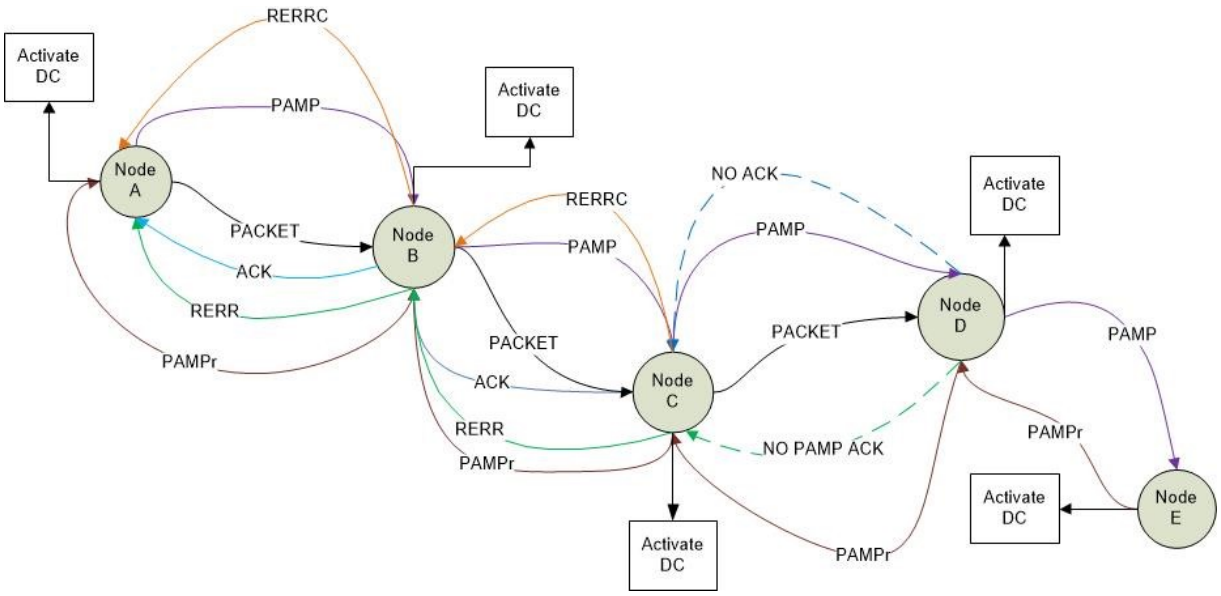

Figure 4-1 Proposed AISBA Model

In the proposed AISBA each node, as seen in Figure. 4-1, is modelled as a DC because DCs are the first line of defence as well as HIS antigen presenting cells. The initiator Node A sends a Route Request to the nodes in the network. The nodes that already have a path to the destination will send back a Route Reply. Upon receipt of the Route Reply, the source node sends its packet to the responder node. During this phase the source node expects the responder node to acknowledge (ACK) packet receipt. In the case of a Route Error causing the ACK to not be received, a Danger Signal alarm is raised upon which the initiator node is notified. This leads to a scenario where the genuineness of the Route Error should be verified, which will be explained in the upcoming sections. DCs in this model act as inquisitors, considering the events around them. They identify the presence of an invader and then present evidence of the invader to T-cells, which in turn activate the appropriate immune cells to attack the intruder.

In a similar fashion the DC nodes, when they do not receive a response from the

neighbouring node, inform the source node and the source node sends the high priority PAMP signal to validate the presence or absence of danger. In our trust model based on AIS, four trust components are considered:

- Safe Signal 1 (SS1) - This is generated upon receipt of Route Reply
- Safe Signal 2 (SS2) - This is generated upon receipt of an ACK
- Danger Signal (DS) - Generated in case of route discrepancies i.e. Route Error (RERR)
- PAMP - This signal helps validate the selfish behaviour of a node. PAMP activates the immune response, thereby protecting the host from infections in HIS. In a similar way PAMP, being a high priority signal, overwrites the node buffer, and the attacker node will acknowledge receipt of PAMP.

The trust value $T_{i,j}^{TC}(t)$ is evaluated by Node *i* towards Node *j* at time *t*, *TC* is the trust component. $T_{i,j}^{TC}(t)$ is represented as a real number in the range of [0, 1] as seen in Figure 4-2 where 1 indicates genuine/unselfish or normalcy of nodes, [0.5-0.8] indicates route error discrepancies and [< 0.5] indicates selfishness.

$$T_{i,j}^{TC}(t) = w_1 T_{i,j}^{SS1} + w_2 T_{i,j}^{SS2} + w_3 T_{i,j}^{DS} + w_4 T_{i,j}^{PAMP} \quad (4\text{-}1)$$

Where w_1, w_2, w_3, w_4 are the weights related to the trust components, with $w_1+w_2+w_3+w_4=1$. Instead of assigning individual weights to each of the trust elements a priority signal, PAMP, is used and a signal, SAFE, to indicate the nodes are behaving correctly. The weight of the PAMP priority signal is shown by w_{PAMP}. The weight of the safe

signal is shown by w_{SAFE}. (4-1) can be rewritten as:

$$T_{i,j}(t) = w_{PAMP}[T_{i,j}^{DS} + T_{i,j}^{PAMP}] + w_{SAFE}[T_{i,j}^{SS1} + T_{i,j}^{SS2}] \quad (4\text{-}2)$$

The values of the weights are chosen to maximize the performance of the algorithm based on Trust models which are evaluated in later sections.

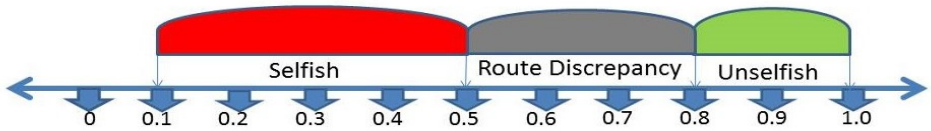

Figure 4-2 Trust Condition Number line model

4.3.1 Trust Condition Evaluation

The calculation of trust at each node is an indicator of the confidence in the node reliability [23]. The trust associated with a node should not be affected by network traffic, congestion and delay. The timing information of each node interaction should not be strictly emphasized.

The use of a sliding window transmission approach reduces the effect of conditions arising out of a network that affect the trust calculation. In most real-time communication scenarios, utilizing sliding window mechanisms do not cause any delay in real-time packet delivery. We use a timing window Δt to evaluate the number of successful and unsuccessful messages between nodes.

Let us consider Node *i* to evaluate Node *j* based on its behaviour; thereby making Node *i* the settlor and Node *j* the trustee.

$$T_{i,j}^{TC}(t) = T_{i,j}^{TC}(t - \Delta t) + T_{i,j}^{TC}(t) \quad (4\text{-}3)$$

To calculate the trust threshold based on Trust conditions (T^{TC}) between two nodes i and j:

$$T_{i,j}^{TC} = T_{ij}^{TC}(t).\,Pcom + .\,e^{-\rho\Delta t}.\,T_{ij}^{TC}(t-\Delta t).\,Pcom \quad (4\text{-}4)$$

The trust relationship between nodes *l, i* and *j* as shown in Figure 4-3 is given by *(l, j) = (l, i): (i, j)*

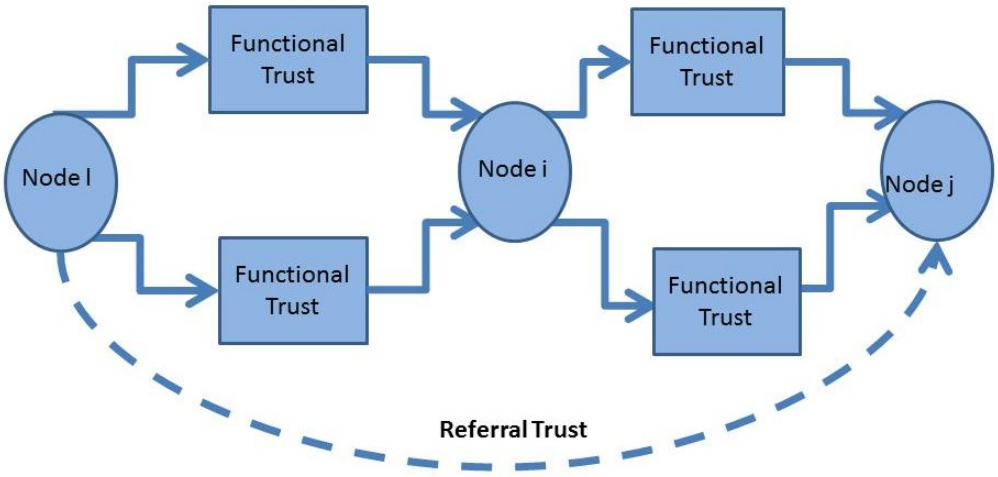

Figure 4-3 Trust Model

Let the Trust Purpose be defined as "the node should be genuine." Let the trust model, as seen in Figure 4-3, include node *i* in the network. The trust between node *l* and node *i* will be direct therefore it's a functional level of trust whereas the trust between node *l* and node *j* will be indirect therefore it's a referral level [24] of trust.

$$T_{l,i,j}^{TC}(t) = T_{l,i}^{TC}(t).\,Pcom + e^{-\rho\Delta t}.\,T_{l,i}^{TC}(t-\Delta t).\,Pcom + T_{i,j}^{TC} + T_{l,j}^{TC}(t).\,Pcom + e^{-\rho\Delta t}.\,T_{l,j}^{TC}(t-\Delta t).\,Pcom \quad (4\text{-}5)$$

(4-5) can be rewritten as

$$T_{l,i,j}^{TC}(t) = T_{l,i}^{TC} + T_{i,j}^{TC} + T_{l,j}^{TC} \quad (4\text{-}6)$$

To compute $T_{i,j}^{TC}$, we take into account the number of interactions between nodes *i* and *j* over the maximum possible number of interactions that could occur with any neighbour node during the interval [0,t]. We consider the following interaction types with regards to a

genuine /unselfish node, given that node i is the initiating node:

- Sending Request
- Receiving Reply
- Selection of node based on highest value of P_{com}
- Acknowledgment
- PAMP signal (high priority signal)

The source node sends a request packet to the other nodes in the network, the nodes which are closer to the destination node will reply. On receiving a response, the source node selects the nodes with the highest value of P_{com}. Once the packet has been forwarded to the node with the highest value of P_{com}, the node, in turn, waits for an acknowledgment from the corresponding node. In the instance of a selfish node, it will not transmit a reply. At this juncture, the high priority PAMP signal plays a critical role. The node that did not receive a response from its neighbour node informs the source node, which in turn leads to "Activate DC" mode being switched on.

The initiator node then sends a PAMP signal ($PAMP_{send}$) and each node is required to acknowledge receipt of $PAMP_{send}$ by sending back a PAMP receive signal ($PAMP_{recv}$). The selfish node that did not formerly acknowledge receipt of the packet will be forced to respond with a $PAMP_{recv}$ as the PAMP signal is a high priority message. The gist of the PAMP signal strength on Packet loss ratio, in the presence of selfish nodes can be viewed in Figure. 4-4. Let the average number of messages from Node i, with a selfish, discrepant (uncertain) and normal node be x, y & z respectively.

If Node *i* requests a neighbour to forward a packet then the messages between Node *i* and selfish Node *j* include reply, selection, acknowledgment, and PAMP signal (total of four) as denoted by x. The anticipated number of messages between Node *i* and a selfish node can be none as there is a probability that the packets will be dropped or forwarded which in turn leads to two (reply, acknowledgment) follow-up messages as denoted by y. In the case of an ordinary node as denoted by z, there can be two classes of communication, including where Node *j* can be an intermediate node forwarding packets or Node *j* can be an unselfish node that replies to a route request from Node *i*.

$$x = P_{com} \times 4 \tag{4-7}$$

$$y = 0 + P_{com} \times 2$$

$$z = P_{com} \times 2 + P_{com} \times 3 = z_1 + z_2$$

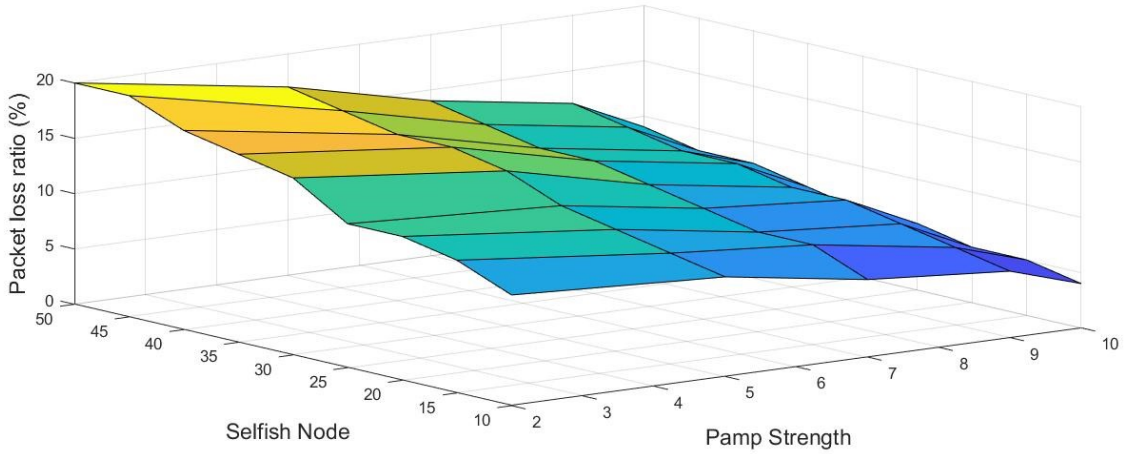

Figure 4-4 Effect of PAMP strength on packet loss ratio

Table 4-1 Trust Component Value Assignments

$T_{ab}^{T.C}(t)$	Values

$T_{i,j}^{PAMP}$	x/z
$T_{i,j}^{DS}$	y/z *1/3
$T_{i,j}^{SS1}, T_{i,j}^{SS2}$	$z_1/z, z_2/z$

Consequently, compute $T_{ab}^{T,C}(t)$ as seen in Table 4-1, by assigning a status value of x/z to selfish states in Node *j*, y/z for a discrepant node and z/z for a normal node. Once Node *a* obtains T_{ab}^{TC} for *TC* = Safe Signal1, Safe Signal2, Danger Signal, PAMP then $T_{ab}(t)$ is calculated based on (4-2).

- $T_{ab}^{SS1}(t)$ This measures the number of times a trustee node generated a route reply. Here a settlor node evaluates the unselfish and honest behaviour of the trustee node. This trust factor is calculated based on the number of interactions between the settlor and trustee node.

- $T_{ab}^{SS2}(t)$ This trust element is evaluated when the trustee node sends back an acknowledgment of receipt of a packet.

- $T_{ab}^{DS}(t)$ In this case, the analysis is done by snooping on the packet transmission activity of the trustee node.

- $T_{ab}^{PAMP}(t)$ This is analysed by observing if the susceptible node has acknowledged the PAMP signal.

4.3.2 Trust Threshold based on Interactions

The Trust conditions are necessary to calculate the node reliability confidence. Traffic issues, such as congestion and delay, should not be taken into account when the trust is to be calculated. In other words, the trust evaluation should not rely exclusively on the network traffic conditions. The sliding window mechanism is applied as it sees the relative transmission time and downplays the consequence of degraded network traffic. The sliding window approach is employed to assess the number of successful and unsuccessful messages between the nodes and the timing windows (t for message transmission are recorded. After a time unit has passed the window slides by one increment and discards the contents of the previous time unit with no further action unless the ACK for the message transmitted in that time increment has not been received. Based on the analysis of the network scenario the window time duration can be varied. Let us consider an example, as depicted in Figure. 4-5, where the time for the window varies.

The timing window length can be a variable and is determined based on the network scenarios. After a time unit has passed the window slides by one increment and discards the contents of the previous time unit with no further action unless the ACK for the message transmitted in that time increment has not been received. Based on the analysis of the network scenario the window time duration can be varied. Let us consider an example, as depicted in Figure. 4-5, where the time for the window varies. In the below given example, the length of the sliding window is three.

During the first-time unit $\Delta t1$, the number of efficient and inefficient interactions is 4 and 3

respectively, and during the entire $\Delta t1$, the total number of efficient and inefficient interactions are 12 (sum of 4, 2 and 6) and 16 (sum 3, 5 and 8) respectively. Once the transition from the first-time unit is completed the new time interval $\Delta t2$ is actioned. In this case, $\Delta t1$ values (4, 3) are discarded and algorithm will be considering the next three values (2, 5), (6, 8), and (7, 1).

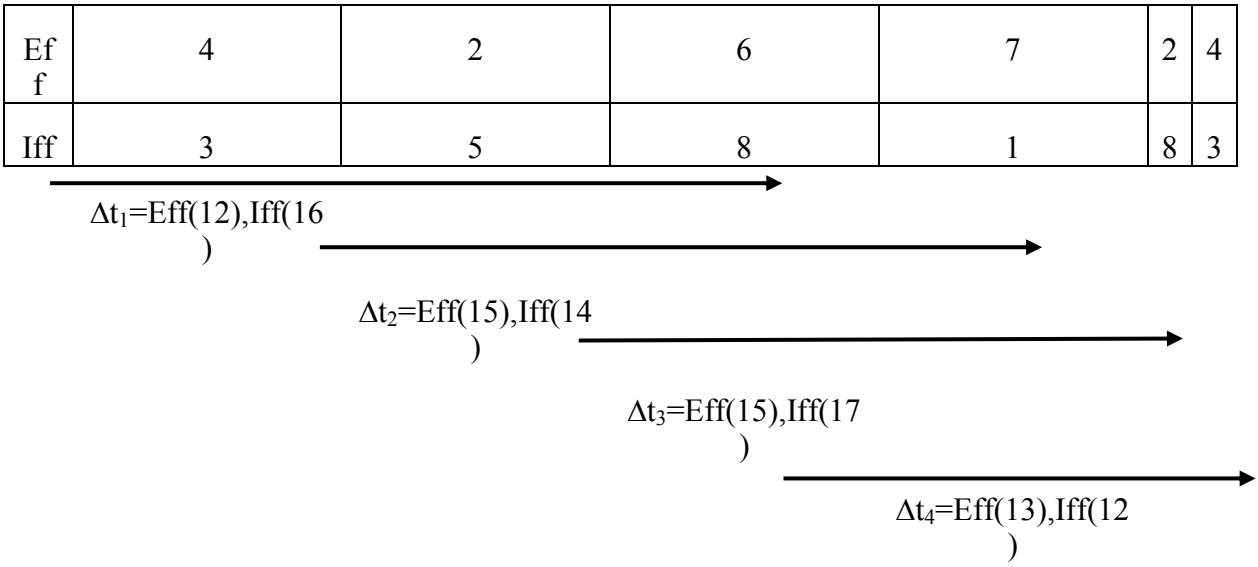

Figure 4-5 AISBA Sliding Window Implementation

With the information gathered from the sliding window mechanism the time-based Trust threshold value of Node j at Node i ($Th_{i,j}$) that lies between 0 and 100 is defined as

$$\mathrm{Th_{i,j}} = \frac{\left(\mathrm{Eff_{i,j}}\right)^2}{|\mathrm{Eff_{i,j}} + \mathrm{Iff_{i,j}}|} \tag{4-8}$$

$Eff_{i,j}$ is the total number of effective interactions of Node i with Node j during Δt. $Iff_{i,j}$ is the total number of ineffective interactions of Node i with Node j during Δt. Equation (4-8) is designed in such a way that the Trust threshold would approach 1 very slowly with the increase in effective interactions; therefore, it would take a longer time for Node i to increase

its trust value for another Node *j*.

Figure. 4-6 shows the variation of Trust threshold value against effective and ineffective interactions. When the number of effective interactions is nil, then the Trust threshold value will be zero.

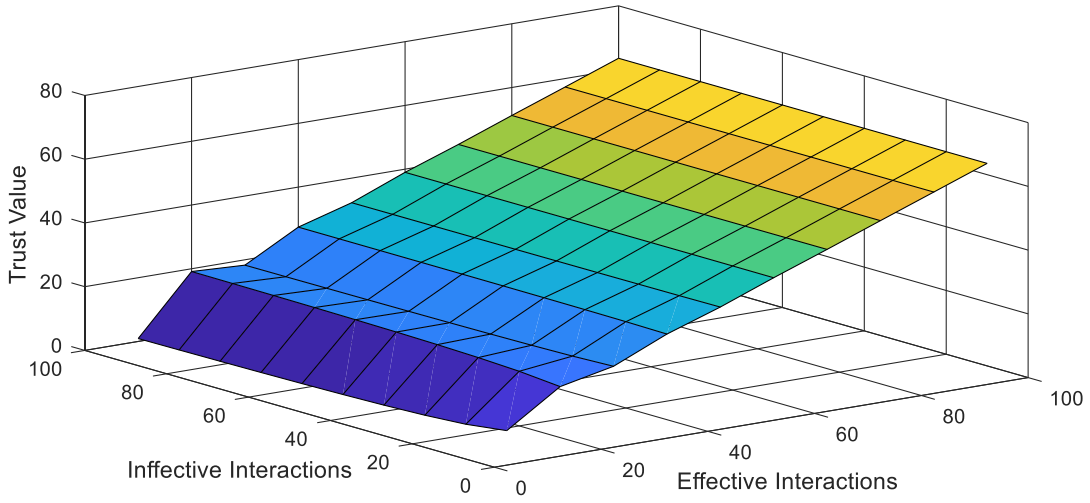

Figure 4-6 Effect of interactions on Trust Value

Based on the calculations of the Trust threshold values as seen in Figure. 4-6, a node classifies the behavioural state (B_{st}) of a node as shown in (4-9) with *a* denoting the median of all the Trust threshold values contributed to by unselfish nodes, and *b* indicates the median of all the Trust threshold values contributed to by the selfish nodes.

$$\mathrm{B.st}\left(\mathrm{Th_{i,j}}\right) = \begin{cases} \text{Unselfish sector}(\text{Safe Signal1}, \text{Safe Signal2}) & 100 > 100 - \mathrm{a} > \mathrm{Th_{i,j}} \\ \text{Unsure sector}(\text{Danger Signal}) & 100 - \mathrm{a} > 50 - \mathrm{b} > \mathrm{Th_{i,j}} \\ \text{Selfish}(\text{PAMP}) & 50 - \mathrm{b} > \mathrm{Th_{i,j}} > 0 \end{cases} \quad (4\text{-}9)$$

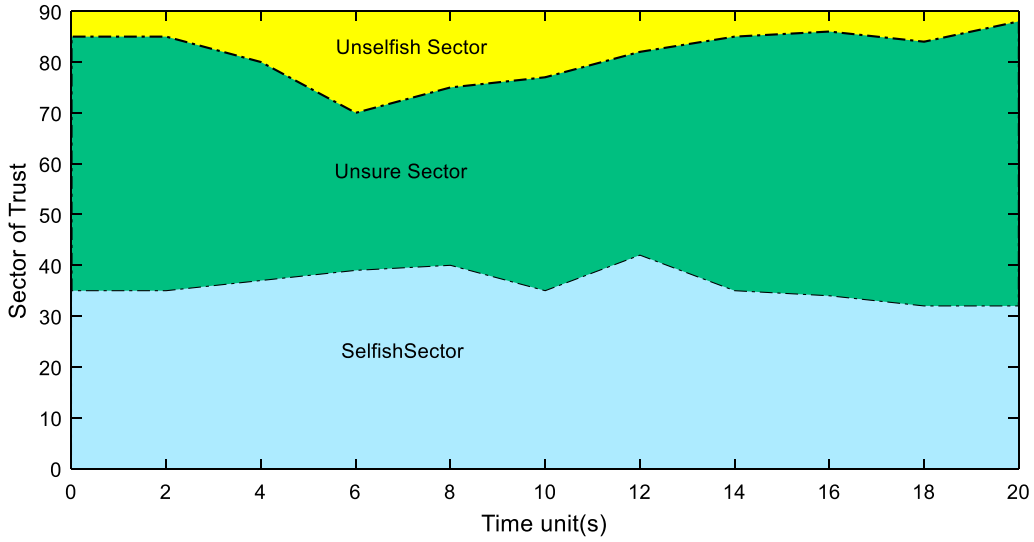

Figure 4-7 Node Behavioral Sectors

The median value provides a typical outcome if a set of values includes an outlier, which is an extreme value that differs largely from other values. The median is not significantly affected by extremely large or small values, so it is used to provide a typical Trust threshold value in a MANET that consists of selfish and unselfish nodes.

In a scenario where there are more altruistic nodes and a low number of selfish nodes the mean value could be high as it would include the Trust threshold value of the unselfish nodes. Both *a* and *b* are calculated as shown in (4-10) and (4-11).

$$a_{j+1} = \begin{Bmatrix} \frac{1}{2}\left(\frac{\widetilde{Th}_{I,J}}{M}\right) & 0 < M \leq n-1 \\ a_j & M = 0 \end{Bmatrix} \quad (4\text{-}10)$$

$$b_{j+1} = \begin{Bmatrix} \frac{1}{3}\left(\frac{\widetilde{Th}_{I,J}}{P}\right) & 0 < P \leq n-1 \\ b_{j+1} & P = 0 \end{Bmatrix} \quad (4\text{-}11)$$

M denotes the set of unselfish nodes; *P* is the set of selfish nodes and *n* is the total number of nodes containing selfish, unselfish and unsure nodes. The values of *a* and *b* are designed to be

robust and can change over time, as shown in Figure. 4-7, hence the boundaries for selfishness or unselfishness can be dynamic. In the event of M or P becoming 0 then the values of a_{j+1}, b_{j+1} will be the same as the previous values a_j, b_j. For the nodes with values over 100, a will represent unselfish nodes, whereas for the nodes with values below 50, b will represent selfish nodes. The Trust threshold value calculation will continue in this manner after every unit of time.

4.4 Simulation and Results

The simulations were done using Network Simulator-3 version 3.23. The challenge was to implement AISBA within the simulation environment. In the simulations, the nodes were modelled as DCs and selfish nodes were introduced to provide an understanding of the algorithm's effectiveness. As observed in Figure. 4-8, using the packet delivery ratio metric, the algorithm performance was evaluated and AISBA provided a packet delivery ratio of 86.25%, while SAODV (Secure AODV) was found to average 36.45%. The performance improvement found using AISBA, with this scenario when compared to SAODV, highlights the potential for AIS algorithms to be effectively utilized in MANET. The packet delivery ratio benefited from the detection and retained knowledge of malicious nodes.

Table 4-2 Simulated Parameters

Simulator	Ns-3.23
Mobility Model	Random waypoint
Simulation Time	950s
Number of nodes	150
Traffic Type	UDP
Network Area	1500m*1500m
No.of malicious nodes	10-50
Mobility	20 m/s
Transmission Range	50m

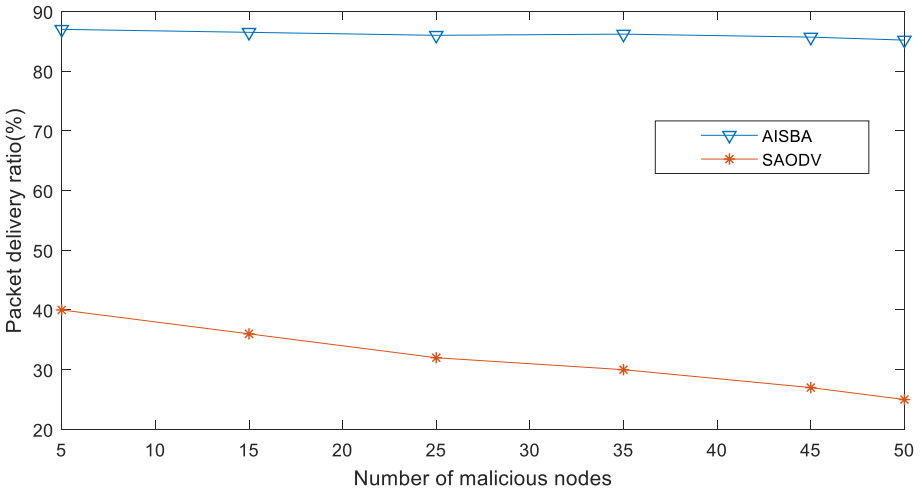

Figure 4-8 Malicious Node Effect on Packet Delivery

Figure. 4-9 shows the AISBA detection rate against the number of malicious nodes in the network. An average detection rate of 93.41% was achieved while for SAODV the detection rate achieved was 85.34%. This comparison demonstrates that as the number of malicious nodes increases, the DC nodes can identify and validate a valid route error as well as a purposeful packet drop caused by the selfish nodes with the avail of the PAMP signal.

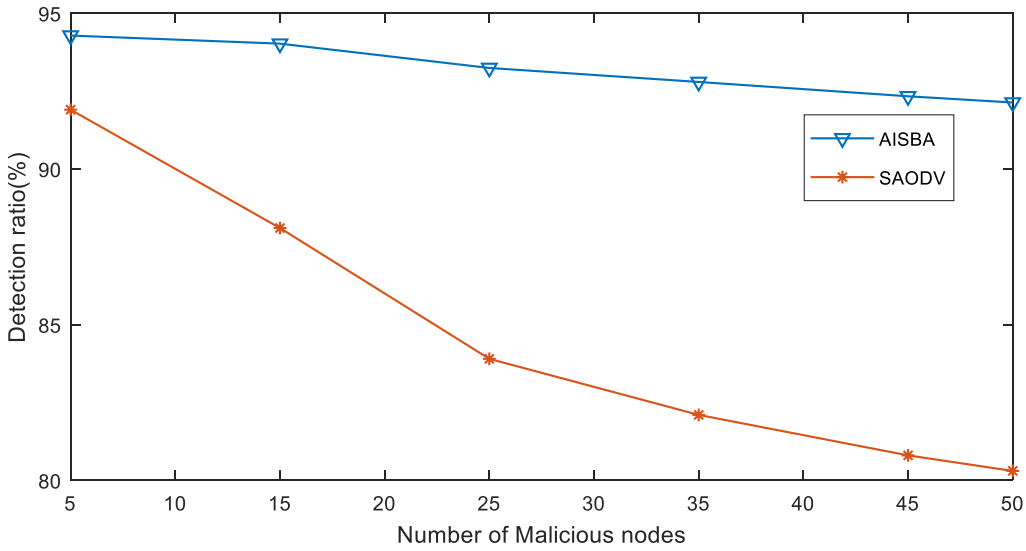

Figure 4-9 Malicious Node Detection Rate

Using the Packet delivery ratio metric as shown in Figure. 4-10 we can evaluate the performance of the proposed security algorithm. AISBA shows higher packet delivery ratio of 86.25%, while SAODV exhibits an average of 36.45%. The contribution of this paper is that the use of bio-inspired algorithms gives better performance compared to SAODV. The packet delivery ratio and detection rate are better even in the presence of malicious nodes.

Figure. 4-11 shows the Detection rate of AISBA against the number of malicious nodes. An average of 93.41% of true detection is achieved while Secure AODV reaches 85.34%. AISBA was evaluated against CITA-AODV. CITA-AODV utilizes the concept of learning stage and learns misbehaviour during this stage, AISBA does not utilize a learning stage as nodes in MANET have a dynamic topology therefore the behaviour of the mobile nodes is not constant. This comparison shows that though the number of malicious nodes increases, the DC nodes can identify and validate a valid route error as well as identify malicious nodes.

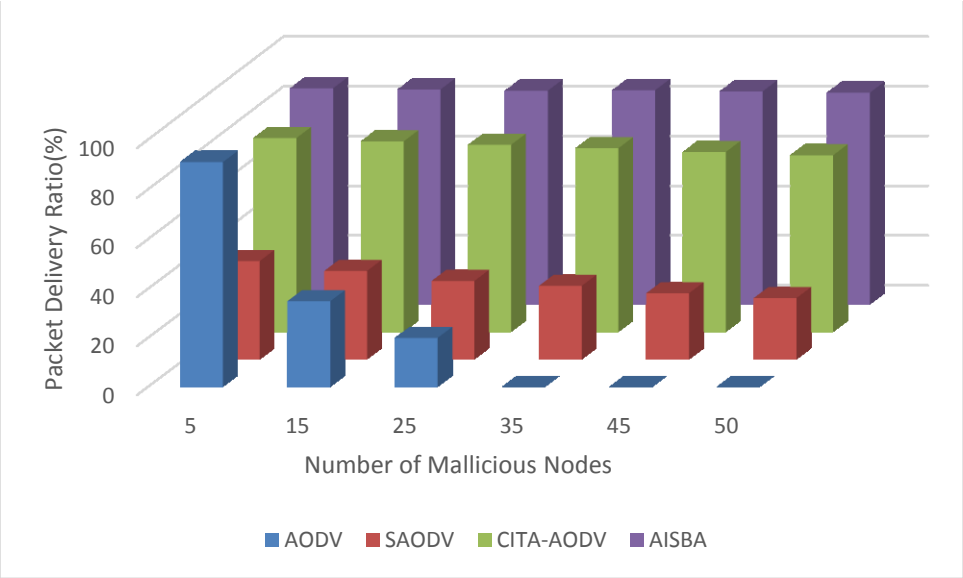

Figure 4-10 Packet Delivery Ratio corresponding to Malicious nodes

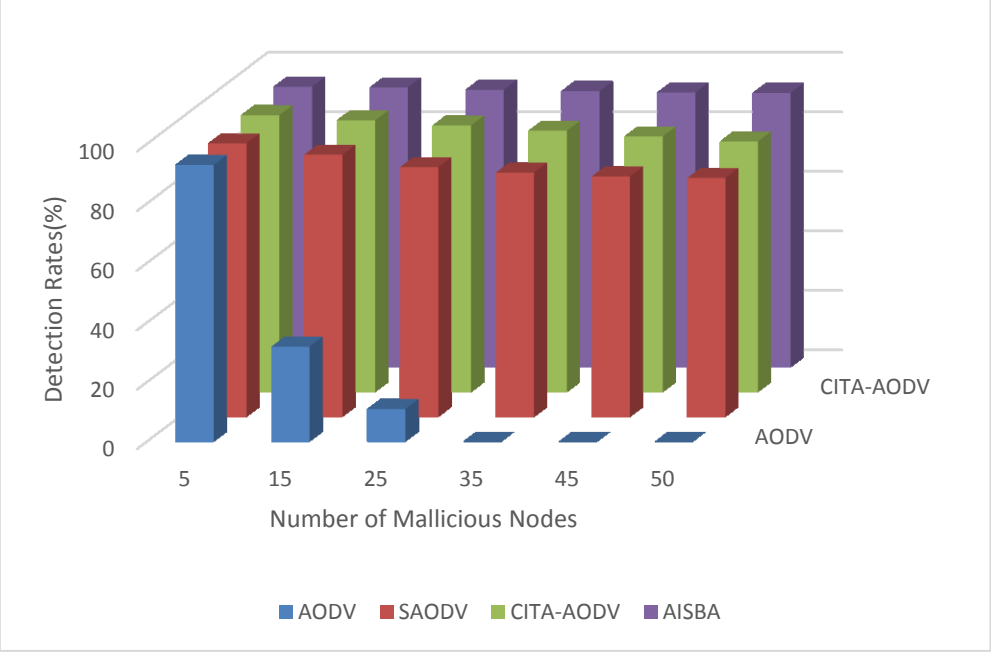

Figure 4-11 True Detection corresponding to Malicious Nodes

4.5 Statistical Analysis

Consider that the trust value towards Node *k* is a random variable *X* following the normal distribution with *n-1* degrees of freedom. This random variable is related to the sample mean, population mean and the standard deviation as follows:

$$X_{i,j}(t) = \frac{\overline{T_{i,j}^{TC}}(t) - \mu_{i,j}^{TC}(t)}{\frac{\sigma_{i,j}^{TC}(t)}{\sqrt{n}}} \quad (4\text{-}12)$$

Where $\overline{T_{i,j}^{TC}}(t)$ is sample mean, $\mu_{i,j}^{TC}(t)$ population mean and $\sigma_{i,j}^{TC}(t)$ the standard deviation of Node *j* as observed by Node *i*. The probability that Node *j* is identified as a selfish node at time *t* is

$$\phi_{i,j}(t) = P(\mu_{i,j}^{TC}(t) < Th_{i,j}) = P\left(X_{i,j}(t) > \frac{\overline{T_{i,j}^{TC}}(t) - Th_{i,j}}{\frac{\sigma_{i,j}^{TC}(t)}{\sqrt{n}}}\right) \quad (4\text{-}13)$$

The analysis of the false positive probability is done by calculating $\phi_{i,j}(t)$ under the scenario

that node *j* is an altruistic/unselfish node whereby the false negative probability is analysed by calculating *(1- $\phi_{i,j}$ (t))* when node *j* is selfish or compromised.

$$\text{fpp} = \text{P}\left(\text{X}_{\text{i,j}}(\text{t}) > \frac{\overline{\text{T}_{\text{i,j}}^{\text{TC}}}\,(\text{t}) \;-\; \text{Th}_{\text{i,j}}}{\frac{\sigma_{\text{i,j}}^{\text{TC}}(\text{t})}{\sqrt{\text{n}}}} \right) \tag{4-14}$$

During calculation of false positive probability the $T_{i,j}^{DS}$= 0 and $T_{i,j}^{PAMP}$ = 0, There is neither danger nor PAMP signals involved as the nodes are genuine. Therefore, (4-14) is rewritten as

$$\text{fpp} = \text{P}\left(\text{X}_{\text{i,j}}(\text{t}) > \frac{\text{w}_{\text{safe}}\,(\,\overline{\text{T}_{\text{i,j}}^{\text{SS1}}}\,(\text{t}) + \overline{\text{T}_{\text{i,j}}^{\text{SS2}}}\,(\text{t})) \;-\; \text{Th}_{\text{i,j}}}{\frac{\sigma_{\text{i,j}}^{\text{TC}}(\text{t})}{\sqrt{\text{n}}}} \right) \tag{4-15}$$

Conversely, during calculation of false negative probability *(1- $\phi_{i,j}$ (t))* the safe signals are absent, i.e. $T_{i,j}^{SS1} = 0$ and $T_{i,j}^{SS2} = 0$. Therefore, the false negative probability is given by:

$$\text{fpn} = \text{P}\left(\text{X}_{\text{i,j}}(\text{t}) < \frac{\text{w}_{\text{PAMP}}\,(\,\overline{\text{T}_{\text{i,j}}^{\text{DS}}}\,(\text{t}) + \overline{\text{T}_{\text{i,j}}^{\text{PAMP}}}\,(\text{t})) \;-\; \text{Th}_{\text{i,j}}}{\frac{\sigma_{\text{i,j}}^{\text{TC}}(\text{t})}{\sqrt{\text{n}}}} \right) \tag{4-16}$$

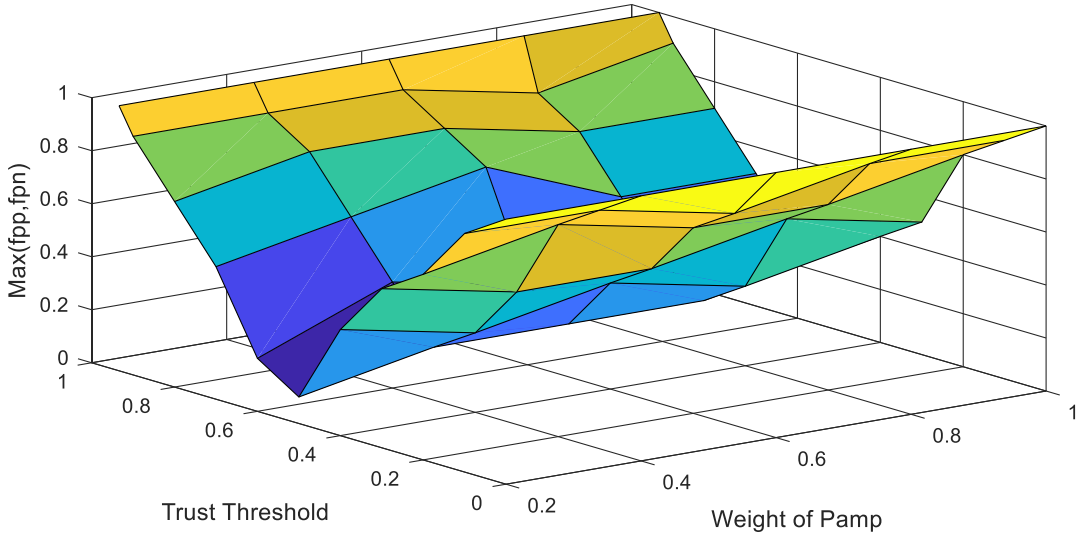

Figure 4-12 Effect of Weight of PAMP and Trust Threshold on max (*fpp,fpn*)

While developing this AIS based detection algorithm, the significance of keeping a balance was kept in mind. We observe that for an absolute value of $Th_{i,j}$, the false positive probability and the false negative probability is minimized. The optimal value for the Trust threshold is 0.5 and the value for Weight of PAMP ($w_{PAMP)}$ is 0.6 as seen in Figure 4-12, so that the false positive and false negative probabilities fall below 5%.

4.6 Summary

The algorithm presented in this chapter aims to provide a bio inspired approach to security. This bio centric approach to MANET security makes it an interesting and challenging subject for further research.

Chapter 5 ROUTING ATTACK - PACKET STORAGE TIME

5.1 Overview

In this chapter, a novel MANET routing attack based on a Packet Storage Time (PST) is presented in response to Research Question 3. An attacking node modifies its storage time and thereby does not forward packets to the intended recipient nodes until some point after the delivery would have normally occurred. In the HIS, cells can discern between a range of matters, including foreign body attacks as well as cellular senescence. This chapter demonstrates a technique that uses the AIS, mimicking the strategy of the HIS, to identify the origin of a PST routing attack.

5.2 Proposed attack-packet storage time

Consider a MANET topology as shown in Figure 5-1, where the ad hoc networks challenge to establish a route between the existing nodes is presented. In Figure 5-1, *MN1* is the source node and *MN7* is the destination node. The intermediate nodes are *MN2-MN6*. In this scenario, consider the following available routes:

- *MN1-MN2-MN6-MN7*
- *MN1-MN5-MN7*
- *MN1-MN4-MN3-MN7*

When a route is needed, the source node should take into account the battery power or energy of the participating nodes that provide a route reply.

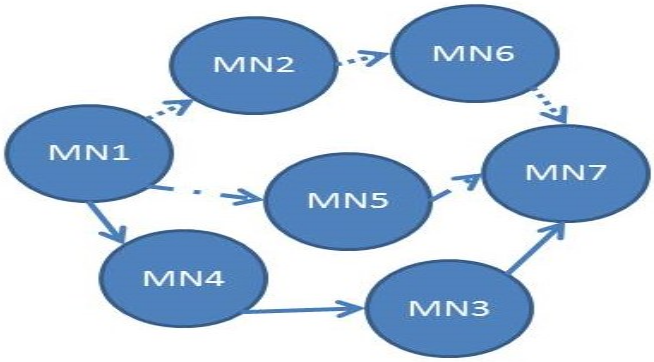

Figure 5-1 MANET scenario

In this scenario, a problem was identified where *MN5* is an attacker/selfish node which do not want to expend energy to forward packets. This node is particularly interested in giving a RREP as it wants to maintain an updated routing table.

The PST attack is a novel concept introduced for the first time in MANET where each mobile node is incorporated with a buffer/queue. In this type of attack, the attacker modifies its own buffer/queue time to congest the network. When the packets are maintained for a longer time than intended by each node, the packets become stale and the circulation of stale packets in the network goes to battery power wastage by the actual nodes.

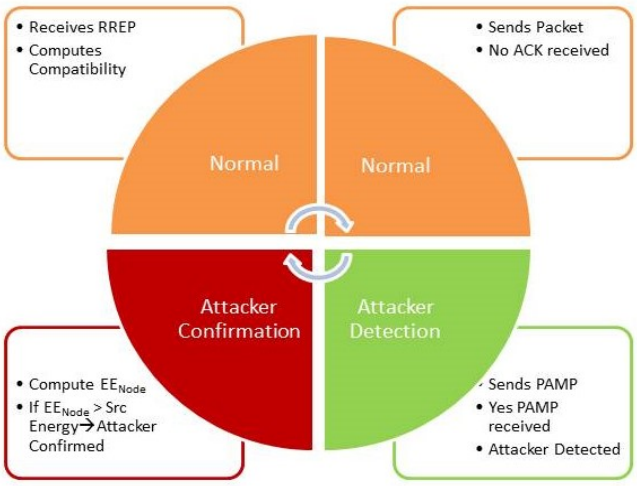

Figure 5-2 Proposed AIS algorithm

5.3 AIS Algorithm

A model based on the Danger Theory principle wherein each node is modelled as a DC is proposed. The presence or absence of danger is detected by the DC nodes, thereby identifying danger by indicating the presence of a malicious node. The DC nodes monitor the activity occurring in a MANET to report any malicious node. The PAMP signal is utilized here to signify the presence of a malicious node in the network. Based on these concepts a mathematical model is built. The nearby nodes have to expend less energy to communicate. The battery life/energy of each DC node plays an important role while establishing routes. During route establishment, the energy of each node needs to be considered. Consider *x* to be an energy dependent variable. The energy associated with the source destination and intermediate nodes is assigned a weight which is dependent on the percentage of battery power that would be used during a route request and route reply communication. Based on the above concept, the below given equation is formulated.

$$f(x) = \alpha n_s + \beta n_d + 2\gamma \bar{n} \quad (5\text{-}1)$$

Where: $\alpha+\beta+\gamma=1$ and $\alpha, \beta, \gamma \in [0, 1]$ and $\bar{n}$ is the average of the energy among intermediate nodes,n_s is the source node energy, n_d is the destination node energy, α is the source node weight factor, β is the destination node weight factor, and γ is the intermediate nodes weight factor. Consider the Effective Energy (EE) of node k

$$\tilde{n}_k = F(n_k, h_k) = EE_{node(k)} = h_k * n_k \quad (5\text{-}2)$$

where h_k is the number of hops from node k to node s, and $F(n_k, h_k) \approx EE_{node(k)}$ should satisfy the postulates:

- If node k is far away from source node s, node k should have to take larger number of hops and more energy would be utilised which results in larger function value.
- If node k is closer to node s, node k should have to take lesser number of hops and lesser energy would be utilised which results in a smaller function value.

The effective mean energy of all the intermediate nodes is as follows

$$\tilde{n} = \frac{1}{m-2} \sum_{k=1, k \neq s,d}^{m} \tilde{n}_k \quad (5\text{-}3)$$

Combining (5-2) and (5-3) gives the Node Energy Momentous function

$$f(x) = \alpha n_s + \beta n_d + \frac{\gamma}{m-2} \sum_{k=1, k \neq s,d}^{m} h_k * n_k \quad (5\text{-}4)$$

From (5-2) we get the Compatibility function

$$\hat{C} = 1/F(n_k, h_k) \quad (5\text{-}5)$$

As Compatibility $\hat{C}$ increases the cost to establish the route between source and destination decreases, which also implies that the node k has energy available for routing. In a MANET,

the source node initiates a route discovery whenever it has to send a packet. The proposed AISBA model consists of the following stages as shown in Figure 5-2:

Normal. Consider the proposed AIS model as shown in Figure 5-2. Initially the source initiates a route discovery in order to send a packet to a destination node and computes compatibility of the node from which it receives a RREP. The source node does not receive an acknowledgement (ACK) from the node that provided the RREP.

Attacker Detection. The source sends a high priority packet PAMP message to the attacker node and the attacker node is forced to acknowledge receipt of the PAMP (PAMP is a high priority signal) which indicates the presence of an attacker but this is not yet confirmed.

Attacker Confirmation. The source computes the node Effective Energy (EE_{node}) of the attacker node and compares the value with its own energy. If the EE_{node} happens to be greater than EE_{source} the presence of the attacker is confirmed.

The algorithm flowchart and pseudo code is described as shown in Figure 5-3. and Table 5-1. The source node broadcasts a RREQ and computes node compatibility for the nodes from which a RREP is received. The source node begins to send the packet and if an ACK is not received a high priority PAMP is sent. If the node is an attacker and it does respond with an ACK; this indicates the presence of the attacker node. The next step taken by the source is to compute the EE of the attacker, and a high EE value is used to confirm the presence of the attacker. This is also symbolically represented in the flowchart as shown in Figure 5-3.

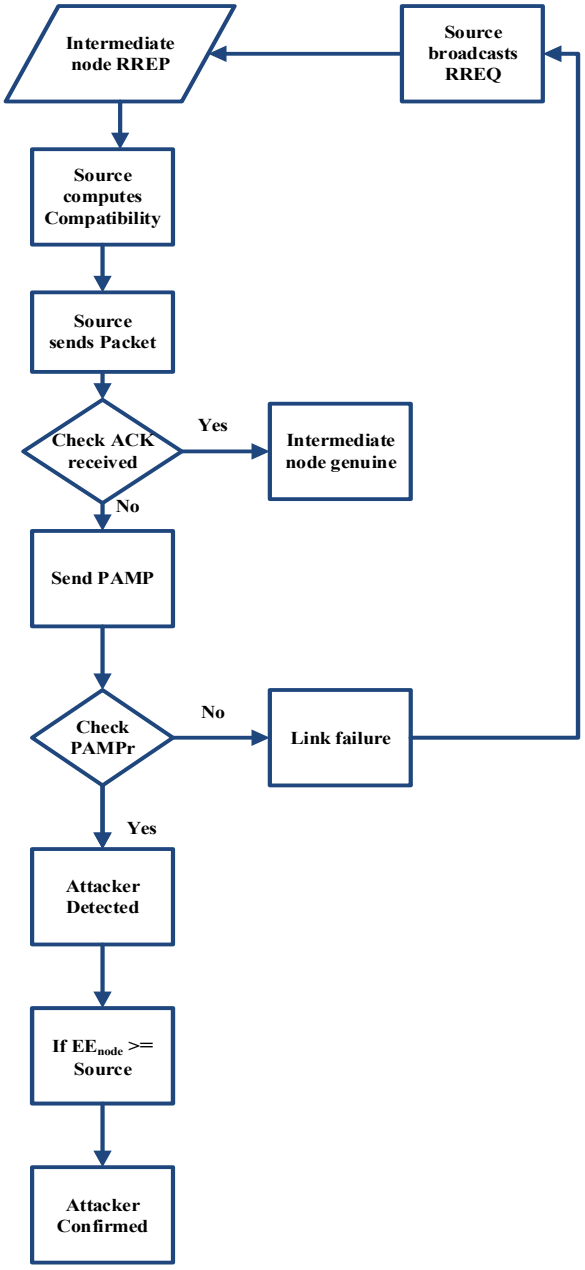

Figure 5-3 Proposed AIS Algorithm flowchart

5.4 Simulation and Results

The ns-3.23 simulator was used to detect and confirm the presence of a PST attacker using the AODV protocol. Extensive simulations were carried out using ns-3.23 to verify the mathematical formulation presented. The simulation parameters used are shown in Table 5-2.

As can be seen in Figure 5-4 as the hop count increases in the network, the EE consumption by the mobile node is higher. As compatibility increases the cost to establish the route between the source and destination will decrease as can be seen in Figure 5-5. The route cost is a metric which is the ratio of transmitted control packets to the transmitted data packets. An increase in compatibility decreases the cost.

Table 5-1 Algorithm Pseudo Code

1. Source Node broadcasts RREQ
(a) $Node_{src}$ broadcasts RREQ
(b) $Node_{intermed}$ sends RREP
2. Compute Compatiblity($Node_{src}$,$Node_{intermed}$,Packet P)
(a) $Node_{src}$ computes compatibility of $Node_{intermed}$;
(b) $Node_{src}$sends packet
(c) $Node_{src}$ does not receive Ack;
3. SendPAMP($Node_{src}$,$Node_{intermed}$,PAMPpacket PAMPp)
(a) $Node_{src}$ sends PAMP,
(b) $Node_{intermed}$ acknowledges PAMP
(c) The presence of attacker detected
4. Compute Effective energy of intermediate node
(a) $EEnode_{(intermed)}$ is computed by Source ,
(b) if $EEnode_{(intermed)}$ >= $Node_{src}$, the presence of attacker is confirmed.
(c) Isolate the attacker node

The average end-to-end delay increases as shown in Figure 5-6 and during the PST the attacker delays the packet delivery by modifying the packet storage time or queue time whereas in an AIS based AISBA model the presence of an attacker is detected and confirmed thereby the packet will be forwarded via a routing path that does not contain the PST

attacker. Hence the average end-to-end delay will be slightly higher for AISBA.

Table 5-2 Simulated Parameters

Simulator	Ns-3.23
Mobility Model	Random waypoint
Simulation Time	500s
Number of nodes	10-50
Traffic Type	UDP
Network Area	600m*600m
Mobility	6 m/s
Pause Time	5s
Transmission Range	50m

As can be seen in Figure 5-7, a PST attack causes packet loss to increase as the number of nodes increase whereas with the AIS based algorithm the packet loss is lower due to the proposed security improvement.

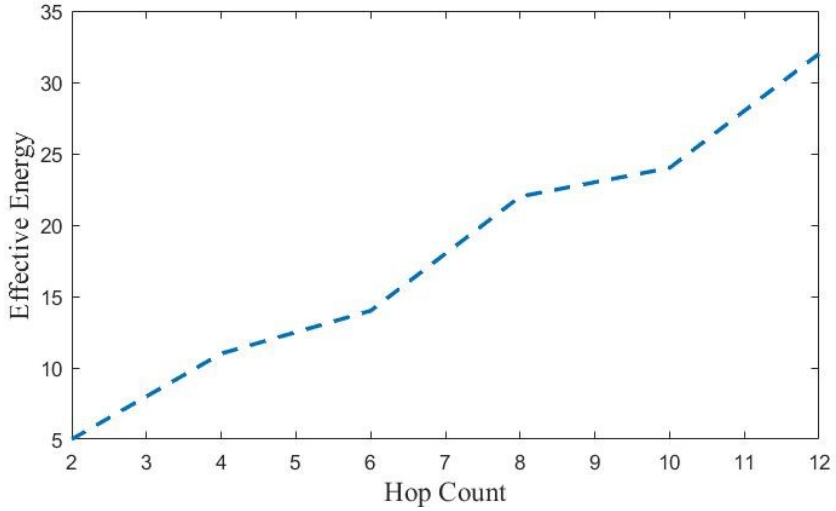

Figure 5-4 Effective Energy v/s Hop Count

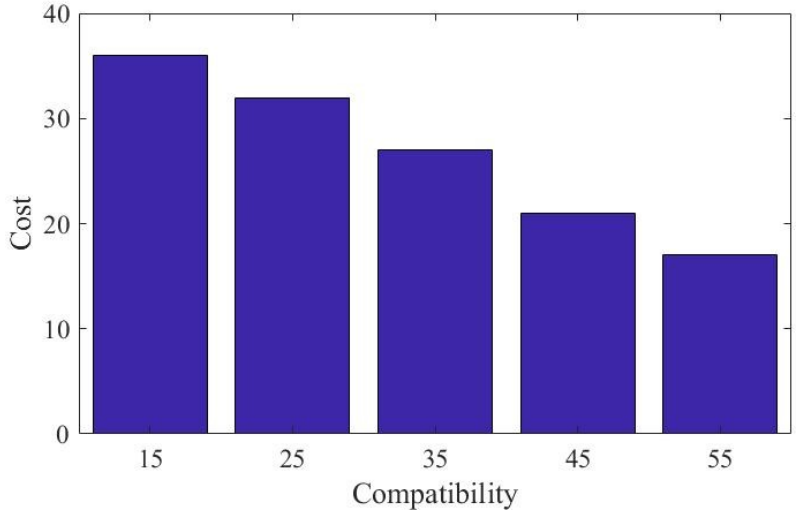

Figure 5-5 Compatibility v/s Cost

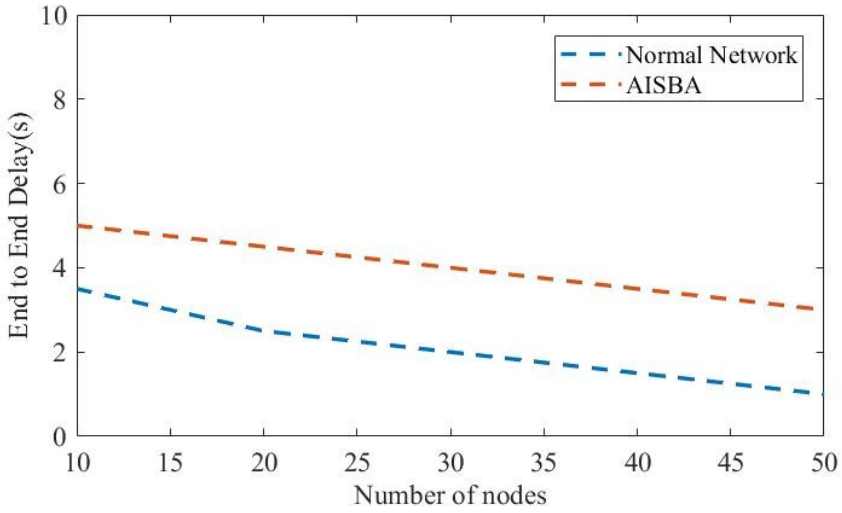

Figure 5-6 E2E v/s Number of nodes

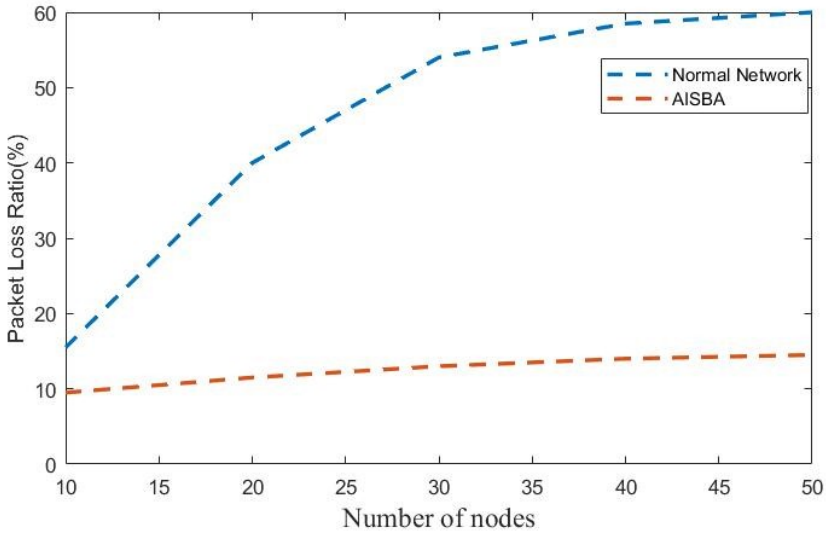

Figure 5-7 PST Attack-Packer loss v/s Number of nodes

5.5 Summary

The novel routing attack presented in this chapter modified the packet storage time in a node buffer to prolong packet delivery and route stale packets in the network. The AIS algorithm developed used three stages to detect and confirm the presence of a PST attacker.

Chapter 6 SELFISH NODE REHABILITATION

6.1 Overview

This chapter provides a description of the Grudger Artificial Immune System framework (GrAIS) in response to Research Question 4. Selfish nodes limit the effectiveness of MANET and it is reasonable in certain situations to adopt an approach that isolates selfish nodes as they're identified or to encourage selfish nodes to switch their behaviour before isolation is imposed. MANET is a communications network that can be utilized for disaster management, military and rescue operations. In each of these scenarios, for MANET to be effective there is a need to limit the number of selfish nodes.

MANET effectiveness is increased when the nodes within the network are active participants, thereby reducing the amount of traffic that is resent due to nodes failing to relay traffic as requested.

This chapter is divided into three sections. Section 6.2 describes the proposed GrAIS. In Section 6.3, simulation results for scenarios with different mission critical workloads are presented. Lastly, Section 6.4 concludes the chapter.

6.2 Modeling of Grudger Artificial Immune System Algorithm

The motivation for the GrAIS framework stems from the observation that it is not beneficial to the operation of a MANET to ignore or isolate selfish nodes. Initially all nodes in the network have the same classification and over time some nodes tend to become selfish. One of the reasons for nodes to become selfish is due to the relay load that the node may have experienced. Traffic workload has a direct effect on energy consumption and as energy reduces nodes can become selfish for various reasons, including observation of the number

and state of neighbouring nodes. The good nodes tend to overlook selfish nodes and continue to render service to the selfish nodes irrespective of any service in return. The proposal is that for high traffic volumes the routing task should be carried out by all nodes, including selfish nodes. In Section 2.4 a model is presented that provided the motivation for the formulation of the GrAIS.

The routing model proposed in this paper categorizes good nodes into a sucker group, cheat nodes into selfish and DC nodes into a grudger group. In the proposed GrAIS model as seen in Figure. 6-1, each node is modelled as a Grudger Dendritic Cell (gDC). This DC node is analogous to the HIS DCs. In HIS, DCs are the first line of defence. The initiator gDC node sends a Route Request to the nodes in the network. The nodes that already have a path to the destination will send back a Route Reply. Upon receipt of the Route Reply, the source gDC node calculates the Pcom of those nodes from which a Route Reply was obtained. The packet is sent to the node that responded with the highest Pcom value.

During this phase, the source node expects nodes to ACK packet receipt. In the case of a selfish node that does not send an ACK, a high priority PAMP signal is raised by the gDC node and the initiator node is also notified. The flow chart of the GrAis model can be seen in Figure 6-2.The selfish node is forced to acknowledge receipt of high priority PAMP signal as this high priority packet overwrites the selfish node's buffer and there upon the packet signal will be transmitted as high priority by the previous gDC node.

In a similar fashion the gDC nodes, when they do not receive a response from the intermediate selfish node, inform the sender node and raises the high priority PAMP signal to

validate the presence or absence of a selfish node. In our AIS based trust model, three trust signals are proposed:

- Safe Signal 1 (SS1) - This is generated upon receipt of Route Reply
- Safe Signal 2 (SS2) - This is generated upon receipt of an ACK
- PAMP - This signal helps validate selfish node behaviour. PAMP activates the immune response, thereby protecting the host from infections in HIS. In a similar way PAMP, being a high priority signal, overwrites the node buffer, and the selfish node will acknowledge receipt of the PAMP.

The trust value $T_{i,j}^{TP}(t)$ is evaluated by Node *i* towards Node *j* at time *t, TP* is the trust purpose. $T_{i,j}^{TP}(t)$ is represented as a real number in the range [0, 1], where 1 indicates unselfish nodes, [0.5-0.8] indicates route error discrepancies and [< 0.5] indicates a selfish behaviour.

$$T_{i,j}^{TP}(t) = w_1 T_{i,j}^{SS1} + w_2 T_{i,j}^{SS2} + w_3 T_{i,j}^{PAMP} \quad (6\text{-}1)$$

Where w_1, w_2, w_3 are the weights related to the trust components, with $w_1+w_2+w_3=1$. Instead of assigning individual weights to each of the trust elements a priority signal, PAMP, is used and a signal, SAFE, to indicate the nodes are behaving correctly. The weight of the PAMP priority signal is shown by w_{PAMP}. The weight of the safe signal is shown by w_{SAFE}. Equation (6-1) can be rewritten as:

$$T_{i,j}(t) = w_{PAMP}[T_{i,j}^{PAMP}] + w_{SAFE}[T_{i,j}^{SS1} + T_{i,j}^{SS2}] \quad (6\text{-}2)$$

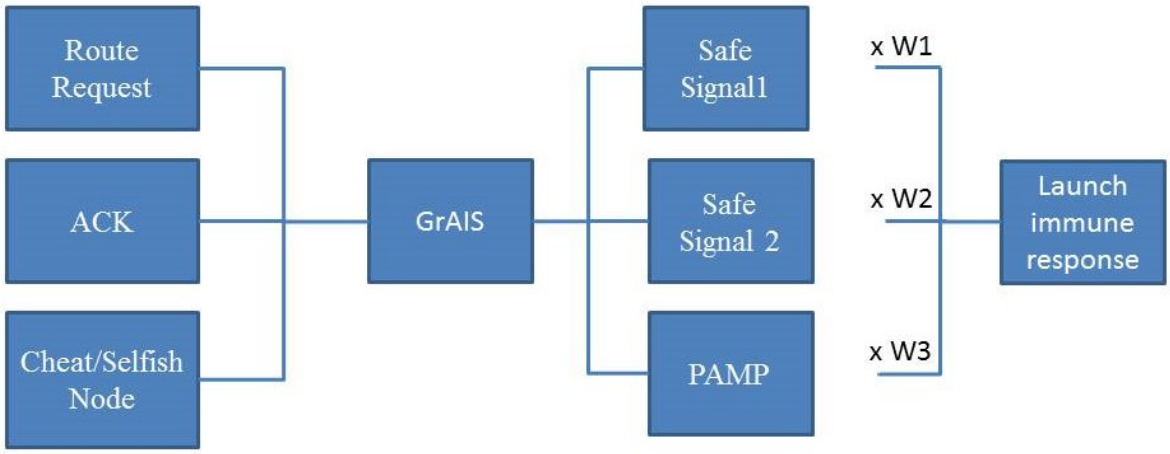

Figure 6-1 GrAIS model
The interaction types between nodes are shown along with the incorporation of the immune response trust model

A sliding window transmission approach is used to decrease the effect of conditions arising out of a network that could affect the trust calculation. A timing window Δt is used to determine the number of successful and unsuccessful packets sent between nodes.

Let us consider a scenario where Node *a* evaluates Node *b* based on its behaviour; thereby making Node *a* trustor and Node *b* the trustee. Node *c* sits beyond Node *b*. The trust relationship between nodes *a*, *b* and *c* is given by (*a*, *b*) = (*a*, *b*): (*b*, *c*).

Let the Trust Purpose be defined as "the node should be good." The trust between Node *a* and Node *b* will be direct therefore it's a functional level of trust whereas the trust between Node *a* and Node *c* will be indirect as well as an exponential decay factor of trust is also considered therefore it's a referral level [24] of trust.

$$T^{TP}_{a,b,c}(t) = T^{TP}_{a,b}(t).Pcom + T^{TP}_{b,c}(t).Pcom + e^{-\rho\Delta t}.T^{TP}_{a,c}Pcom \qquad (6\text{-}3)$$

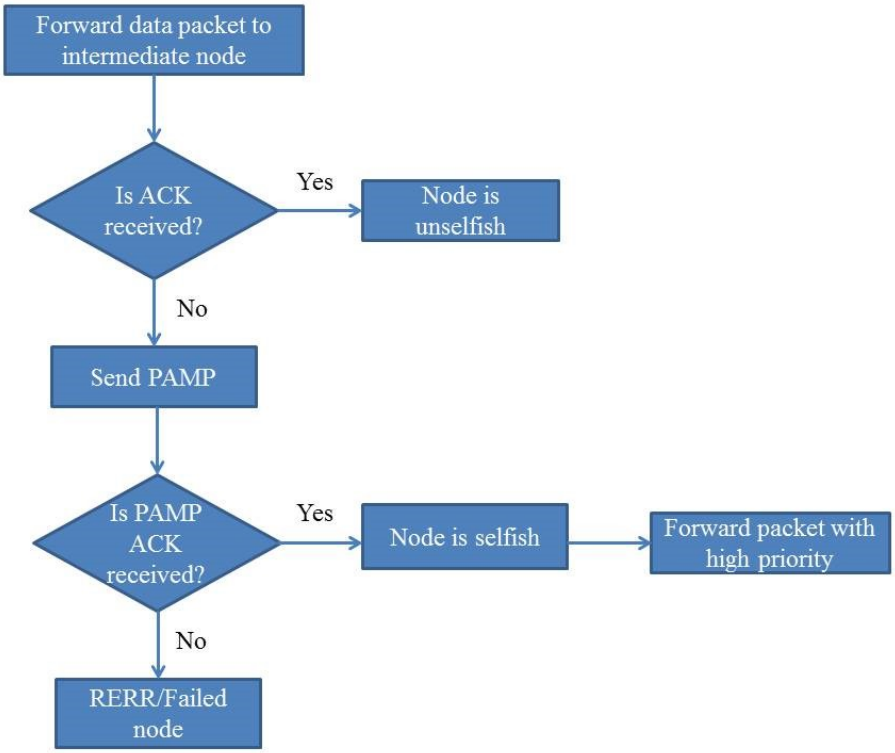

Figure 6-2 GrAIS model event flow

To compute $T_{a,b}^{TP}$, consider the number of interactions between nodes a, b and c over the maximum possible number of interactions that could occur with any neighbor node during the interval [0, t]. The hop count measure calculated by P_{com} and the Effective Energy of each node (EE_{node}) is detrimental during the interaction between nodes in the GrAIS model. In this approach, the following categories of interaction with regards to an unselfish node, given that Node a is the initiating node is considered:

- Sending Request
- Receiving Reply
- Selection of node based on highest value of P_{com}
- Send Packet

- If no ACK, send PAMP
- If PAMP received, classify node as selfish node.
- gDC node will resend packet to selfish node

Pcom is an important factor while evaluating the trust purpose ($T_{a,b}^{TP}(t)$) between any two nodes as the packet will be sent to the node that responds with a route reply and highest P_{com} value. In this approach $T_{a,b}^{TP}(t)$ between any two neighbouring nodes is computed by taking into account the number of communications between nodes a and b over the maximum possible number of interactions that could occur with any neighbour node during the interval [0,t]. The trust purpose for Safe Signal 1 is computed by taking the ratio of the total number of route replies (N_{RREP}) received with the total number of route requests sent (N_{RREQ}). The trust purpose for Safe Signal 2 is computed by taking the ratio of the total number of acknowledgement packets (N_{ACK}) received by the sender with the route reply packets sent by the destination/intermediate (N_{RREP}) node. The trust purpose for the PAMP signal is computed by taking the ratio of the total number of PAMP sent for every route reply received by the sender and no acknowledgement sent by the destination/neighbour node.

$$T_{a,b}^{SS1}(\mathrm{t}) = \left[\frac{N_{RREP}}{N_{RQ}}\right].P_{com} \tag{6-4}$$

$$T_{a,b}^{SS2}(\mathrm{t}) = \left[\frac{N_{ACK}}{N_{RREP}}\right].P_{com} \tag{6-5}$$

$$T_{a,b}^{PAMP}(\mathrm{t}) = \left[\frac{N_{PAMP}}{N_{RREP}}\right].P_{com} \quad (6\text{-}6)$$

The intermediate node informs the source node of a neighbouring node that appears to be selfish. The source node sends a PAMP signal to overwrite the selfish node buffer and this selfish node is added to a blacklist to prevent it being used in future communications if it does not respond to the PAMP signal. The high priority PAMP signal plays a vital role in this process. The "Activate DC" mode that is switched on due to a selfish node being identified sets in motion the response process. The effect of the PAMP signal in the presence of selfish nodes and its impact on packet loss ratio can be seen in Chapter 4. As the PAMP works to deal with the selfish nodes, the packet loss ratio is reduced.

The source node sends a PAMP signal, $PAMP_{send}$ and each node should acknowledge receipt by sending back a PAMP receive signal, $PAMP_{recv}$. The selfish nodes that do not formerly acknowledge receipt of the packet will be forced to respond with a PAMP receive signal, $PAMP_{recv}$, as the PAMP signal is a high priority message and it overwrites the node buffer.

- $T_{a,b}^{SS1}(t)$: Measures the number of times any intermediate (trustee) node generated a route reply. Here a settlor node evaluates the unselfish and honest behavior of the trustee node. This trust component is computed based on the number of interactions between the trustor and trustee node.

- $T_{a,b}^{SS2}(t)$: The trust element is evaluated when the trustee node sends back an acknowledgment of receipt of a packet.

- $T_{a,b}^{PAMP}(t)$: Analysed by observing if the intermediate node received no acknowledgement to the data packet but it did send a route reply earlier and then the PAMP signal is sent to validate selfish node behaviour.

The GrAIS utilizes the concept of a Price of Anarchy [9] for load calculation. Consider that there are N nodes in the network. In the GrAIS model, nodes that perform a routing task employ a trust purpose $T_{a,b}^{TP}$ between any two nodes a, b. The EE_{node} of b and Trust value of Node b as observed by Node a is taken into consideration. Therefore, the Workload (WL) in a routing task undertaken by any Node b is

$$WL_b = \frac{1}{EE_{node(b)} \cdot T_{a,b}^{TP}(t)} \tag{6-7}$$

The workload is dependent on the energy of a node or inversely proportional to node energy and trust. (6-7) shows that as workload increases the nodes expend more energy to carry out networking tasks. As the node energy consumption increases the trust value could reduce.

6.3 Simulation results and analysis

The simulations were carried out using NS-3 and MATLAB. Energy-aware workload [13] distribution is the most efficient approach to reduce energy consumption and stimulate cooperation of selfish nodes. In the traditional MANET applications, the workloads are very simple and wireless communication is usually the most energy intensive process. However, as the MANET applications become more complex, it becomes necessary to efficiently distribute the workloads by considering both the trust, hop count and communication energy consumption. In this research, workload in terms of the trust metric and energy consumption during packet transmission was considered. The workload in terms of packet transmission

was considered to reveal the tradeoff between sucker (good) nodes and selfish (cheat) nodes.

Table 6-1 Simulated Parameters

Simulator	Ns-3.23
Mobility Model	Random waypoint
Simulation Time	1000s
Number of selfish nodes	10-50
Number of nodes	150
Traffic Type	UDP
Network Area	300m*1500m
Packet size	130 bytes
Mobility	20 m/s
Transmission Range	50m

In the simulations, there are three workload scenarios explored with the packet delivery workload increasing from Workload1 to Workload3.

In Figure 6-3 and Figure 6-4, it can be observed that initially good nodes maintain trust while the selfish nodes choose to conserve their energy. As the workload is initially light and all nodes have more energy, the unselfish characteristic amongst participating nodes becomes a crucial factor when determining trust. The prominent drop in trust amongst the good nodes was observed at t = 300 min when the good nodes had depleted their energy to a point where they began to look for alternative pathways that would conserve the energy of known good nodes.

The GrAis model performs better as time progresses due to selfish nodes being forced to co-operate. The selfish node maintains trust for a longer *t* as it would have conserved energy to this point.

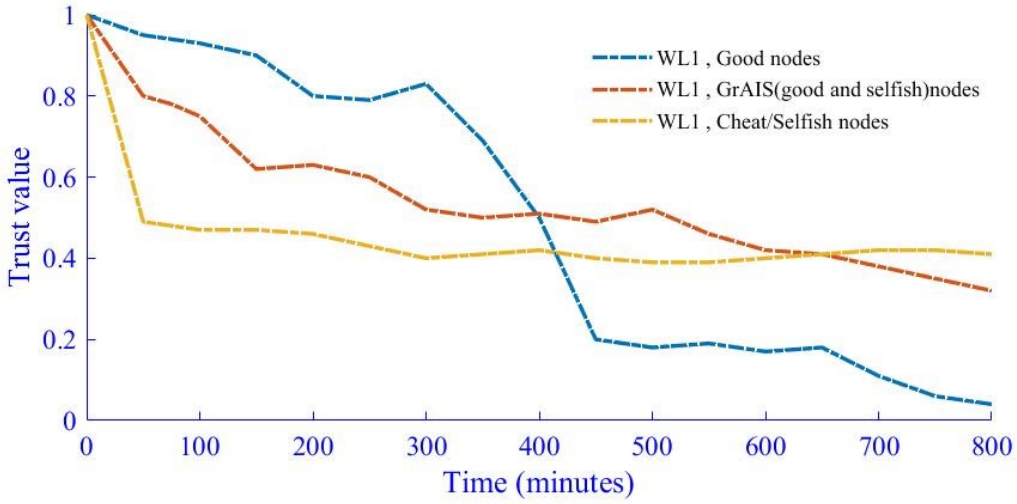

Figure 6-3 Trust values for Workload1

In Figure 6-4, the trend is similar to Figure 6-3, except that the time during which good nodes start to show a dip in trust occurs earlier than when it occurred in the GrAIS model. This is due to the workload increase from Workload1 to Workload2 where the energy consumption increases and good nodes diminish their energy store at a corresponding rate whilst cheat nodes act to retain energy.

The GrAIS model facilitates traffic flows using selfish nodes and as a result the GrAIS model can function more effectively as time progresses when compared to a model that relies upon good nodes to transfer traffic flows.

In Figure 6-5, a new trend is seen with the cheat nodes acting to conserve energy earlier due to the higher workload and this result in a lack of cooperation from the point where trust dips. The GrAIS model approaches the good node model by forcing the selfish nodes to cooperate with the help of the high priority PAMP signal.

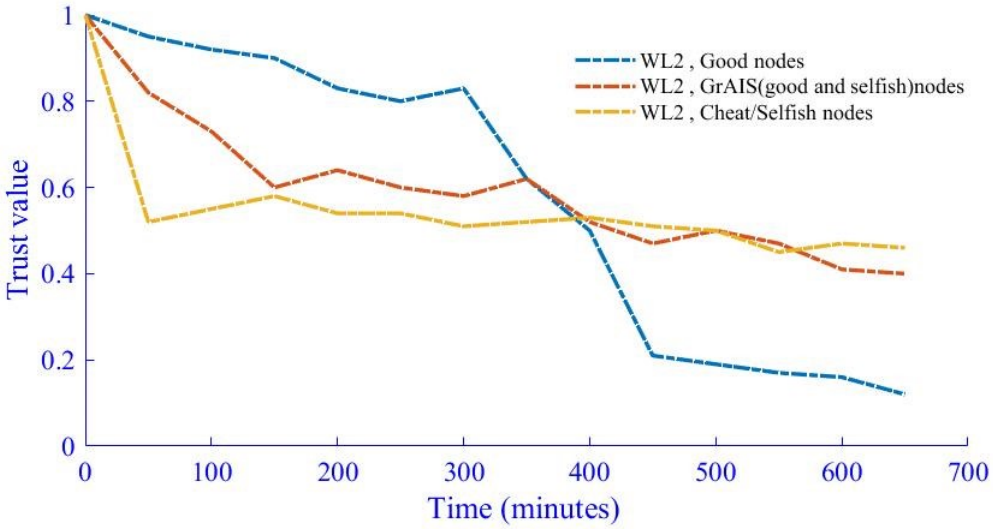

Figure 6-4 Trust values for Workload2

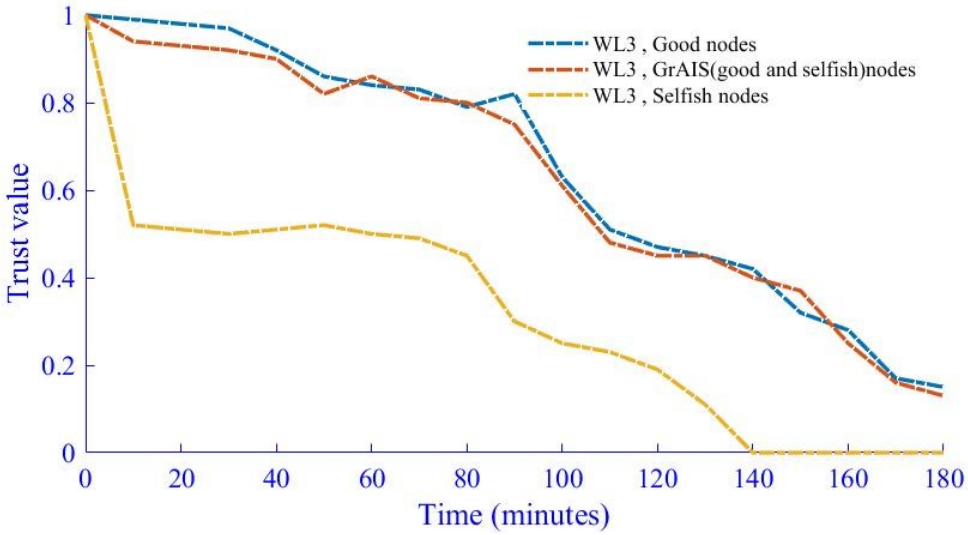

Figure 6-5 Trust values for Workload3

6.4 Summary

The GrAIS model utilizes the principles of AIS and probability to create a model incorporating good and selfish nodes to combat selfishness in MANET. The results obtained from the simulations have shown that the GrAIS model outcomes are an improvement over models that ignore or isolate selfish nodes as time progresses in spite of increasing workload.

A balance between energy utilization, due to good nodes transferring traffic, and energy conservation, due to selfish nodes refusing to transfer traffic, has been achieved by forcing selfish nodes to participate at an appropriate point in the MANET life cycle. A MANET that combines selfishness and unselfishness can be shown to be beneficial when resources, particularly energy, become limited. As future work, a more complex model could be developed exclusively for higher workloads by considering the stability of the GrAIS model over a longer time interval.

Chapter 7 AIS PAIN MODELLING

7.1 Overview

This chapter provides an investigation of AIS pain modelling in response to Research Question 5. The most prominent characteristic of most systems is self-preservation, which is the instinct with which an animal can fight against pain. Most pain is a discomfort but can reach a level that reflects the degree of injury or reflective concern about an event. This discomfort in a MANET can be modelled with the help of key parameters like trust, energy and workload. A MANET node behaves automatically like a sensory node in terms of behavioural patterns like communication and information gathering with other network nodes.

7.2 Modeling Trust for Pain Abstraction

To model the Trust Metric the trust amongst nodes is considered, with the nodes modelled as DCs. When the DC nodes do not receive a response from a neighbouring node they inform the source node and the source node sends the high priority PAMP signal to validate the presence or absence of pain. In the AIS based trust model, four signals for trust purposes are considered:

- Secure Signal 1 (SC1) - This signal is to confirm the receipt of Route Reply.
- Secure Signal 2 (SC2) - This signal is to confirm acknowledgement of data packet.
- Identifier Signal (IS) - This signal is initiated by the intermediate DC node when its immediate neighbour node is the cause of pain in the network. This signal has priority lesser than PAMP as it is the predecessor of PAMP. The signal is produced in the event of route error.

- PAMP - This signal helps confirm the presence of selfish nodes. PAMP is a high priority signal, overwrites the node buffer, and the selfish node (the node that caused pain in the network) will acknowledge receipt of PAMP. This signal is initiated by the source DC node.

The trust value $T_{x,y}^{TP}(t)$ is evaluated by Node *i* towards Node *j* at time *t*, *TP* is the trust purpose. $T_{x,y}^{TP}(t)$ is represented as a real number in range of [0, 1] where 1 indicates good nodes, [0.5-0.8] indicates route error/selfishness and [< 0.5] indicates pain and inflammation.

$$\mathrm{T}_{\mathrm{x,y}}^{\mathrm{TP}}(\mathrm{t}) = \mathrm{w}_1\mathrm{T}_{\mathrm{x,y}}^{\mathrm{SC1}}(\mathrm{t}) + \mathrm{w}_2\mathrm{T}_{\mathrm{x,y}}^{\mathrm{SC2}}(\mathrm{t}) + \mathrm{w}_3\,\mathrm{T}_{\mathrm{x,y}}^{\mathrm{IS}}(\mathrm{t}) + \mathrm{w}_4\mathrm{T}_{\mathrm{x,y}}^{\mathrm{PAMP}}(\mathrm{t}) \qquad (7\text{-}1)$$

Where w_1, w_2, w_3, w_4 are the weights related to the trust components, with $w_1+w_2+w_3+w_4=1$. Instead of assigning individual weights to each of the trust elements a priority signal, PAMP, is used and a signal, SECURE, to indicate the nodes are behaving correctly. The weight of the PAMP priority signal is shown by w_{PAMP}. IS is used to identify route error. The weight of the IS is given by w_{IS}.The weight of the secure signals are shown by w_{SC1} and w_{SC2}.

Equation (7-1) can be rewritten as:

$$\mathrm{T}_{\mathrm{x,y}}^{\mathrm{TP}}(\mathrm{t}) = \mathrm{w}_{\mathrm{PAMP}}[\mathrm{T}_{\mathrm{x,y}}^{\mathrm{PAMP}}(\mathrm{t})] + \mathrm{w}_{\mathrm{IS}}\,[\mathrm{T}_{\mathrm{x,y}}^{\mathrm{IS}}(\mathrm{t})\,] + \mathrm{w}_{\mathrm{SC1}}[\mathrm{T}_{\mathrm{x,y}}^{\mathrm{SC1}}(\mathrm{t})] + \mathrm{w}_{\mathrm{SC2}}[\,\mathrm{T}_{\mathrm{x,y}}^{\mathrm{SC2}}(\mathrm{t})] \qquad (7\text{-}2)$$

The values for the weights are chosen to maximize the performance of the Trust model. The PAMP signal is assigned the highest weight $w_{PAMP} > w_{IS} > w_{SC2} > w_{SC1}$. Where $w_{PAMP} + w_{IS} + w_{SC2} + w_{SC1} = 1$.

The calculation of trust at each node is an indicator of the confidence in node reliability. The trust associated with a node should not be affected by network traffic, congestion and delay.

Let us consider Node *x* to evaluate Node *y* based on its past and present behaviour; thereby making Node *x* the trustor and Node *y* the trustee.

$$T_{x,y}^{TP}(t) = T_{x,y}^{TP}(t - \Delta t) + T_{x,y}^{TP}(t) \tag{7-3}$$

Let the Trust Purpose be defined as the node should be "unselfish." To compute $T_{x,y}^{TP}(t)$ take into account the number of communications between nodes *x* and *y* over the maximum possible number of interactions that could occur with any neighbour node during the interval [0, t]. The trust value for Secure Signal1 is the ratio of the total number of route replies (N_{RP}) received for every total number of route requests sent (N_{RQ}). The trust value for Secure Signal 2 is the ratio of the total number of acknowledgement packets (N_{ACK}) received for every route reply received (N_{RP}).The trust value for the IS is the ratio of the total number of ISs sent for every route reply received and no acknowledgement sent. The trust value for the PAMP signal is the total number of PAMP sent for every IS sent.

$$T_{x,y}^{SC1}(t) = \left[\frac{N_{RP}}{N_{RQ}}\right] \tag{7-4}$$

$$T_{x,y}^{IS}(t) = \left[\frac{N_{IS}}{N_{RP}}\right] \tag{7-5}$$

$$T_{x,y}^{PAMP}(t) = \left[\frac{N_{PAMP}}{N_{IS}}\right] \tag{7-6}$$

The IS is used to identify route error or selfish nodes. When the IS is sent, if there is a genuine case of route error there will not be acknowledgement of the IS and this will be

identified as a possible cause of route error. The intermediate node will inform the source node, which in turn sends a high priority PAMP and this signal overwrites the selfish node buffer and for future communication scenarios this selfish node could be blacklisted.

In the case of a selfish node, it will not send a response. It is, at this juncture, the high priority PAMP signal that plays a vital role. The node that did not receive a response from its neighbour node informs the source node, which in turn leads to "Activate DC" mode being switched on.

The initiator node then sends a PAMP signal, $PAMP_{send}$ and each node is required to acknowledge receipt by sending back a PAMP receive signal, $PAMP_{recv}$. The selfish node that did not formerly acknowledge receipt of the packet will be forced to respond with a $PAMP_{recv}$, as the PAMP signal is a high priority message and it overwrites the node buffer.

Once Node x obtains $T_{x,y}^{TP}$ for *TP*=Secure Signal1, Secure Signal2, Danger Signal, PAMP then $T_{x,y}(t)$ is calculated based on (7-2).

- $T_{x,y}^{SC1}(t)$: This measures the number of times any intermediate (trustee) node generated a route reply. Here a settlor node evaluates the unselfish and honest behavior of the trustee node. This trust component is computed based on the number of interactions between the trustor and trustee node.

- $T_{x,y}^{SC2}(t)$: This trust element is evaluated when the trustee node sends back an acknowledgment of receipt of a packet.

- $T_{x,y}^{IS}(t)$: This signal is sent by the intermediate DC node. In this case, the analysis is done by snooping on the packet transmission activity of the trustee node.
- $T_{x,y}^{PAMP}(t)$: This is analysed by observing if the node was in pain and then then the PAMP signal is sent to validate selfish behaviour in a node.

7.3 Modeling the Pain Reduction Artificial Immune System Algorithm

The basic principle involved in pain modelling is understanding pain as an output of the brain that is produced as a resultant of the body tissue in danger and the need to initiate an action. Pain is never straightforward although it appears to be so. In the Pain Reduction based AIS algorithm, the nodes are analogous to cells in the HIS, and lack of cooperation by any node inflicts pain to the network. There are selfish MANET nodes that appear reluctant to forward packets for other nodes as they do not want to drain their energy.

In this research pain in the network is caused due to selfish nodes as seen in Figure 7-1. Energy consumption is a crucial factor in MANET. Four possible energy consumption states are identified: Sending state (RREQ), Receiving state (RREP), selfish and failed. The first two states are when the node is transmitting and receiving packets respectively, the selfish state has greater effect on energy consumption than the transmitting and receiving states as the node has elected to be selfish to preserve its energy. Nominally the energy cost of a packet is proportional to the packet size. The work presented in [23] identifies an energy consumption model that incorporates the cost of the receiving and sending traffic. The analysis in this approach gives a basis of comparison of the overhead associated with routing as well as data traffic.

The Energy Cost (E_{cost}) associated with each packet at a node is represented as the total of the incremental costs (*c)* proportional to the packet size (*s*) and cost (*d)* associated with channel acquisition.

$$E_{cost} = cs + d(\frac{N_S}{N_T}) \tag{7-7}$$

Where N_S is the number of selfish nodes and N_T is the number of network nodes. When inflammation occurs, chemicals from the body's white blood cells are released into the blood or affected tissues to protect the body from foreign substances. This release of chemicals increases the blood flow to the area of injury or infection, and may result in skin coloration and warmth. Some of the chemicals cause a fluid leak into the tissues, resulting in swelling. This protective process may stimulate nerves and cause pain.

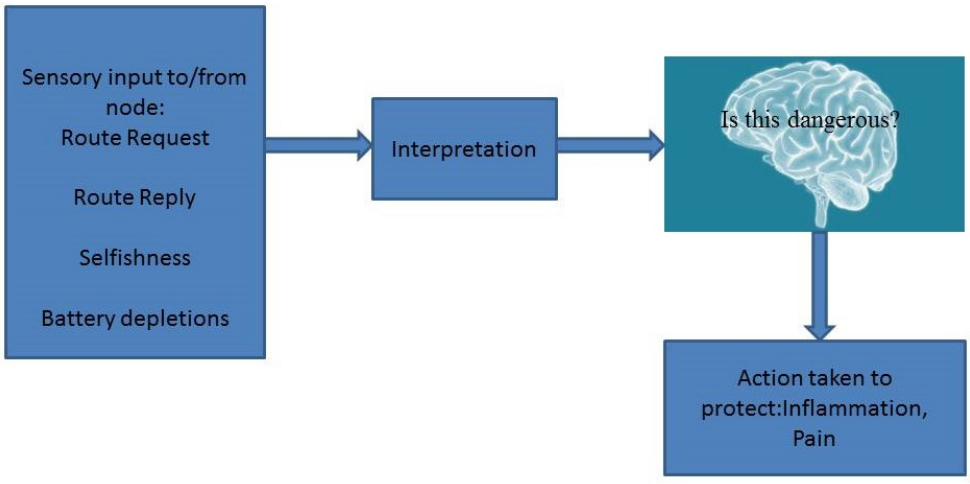

Figure 7-1 Pain and Inflammation Conceptualization in MANET

In MANET during node interaction, the nodes are analogous to DCs in the HIS, when the nodes are stressed there should be a solution to overcome the pain and resulting inflammation. Inflammation (Infl) is initiated upon tissue injury and sets off a cascade of

biochemical reactions that prime the nervous system for pain sensing. Therefore, taking steps to ease inflammation is an effective means of interfering with the process of pain sensitization. Similarly, when there is a decrease in the trust metric an increase in inflammation occurs which indicates the network node is in pain. (7-8) shows the Inflammation-trust relation in AIS based MANET.

$$Infl(t)\ \alpha\ \frac{1}{T_{x,y}^{TP}(t)} \tag{7-8}$$

Pain was modelled using two categories: P_{ba} and P_{aa}. Firstly, the nodes are placed in an environment with selfish nodes, when a node interacts with good nodes the Secure Signals 1&2 are produced due to receipt of Route Reply and acknowledgment of the data packet. In the event of a good node communicating with a selfish node there will neither be receipt of Route Reply nor acknowledgment and during this time the good node is in pain as it affects the performance of the network as well.

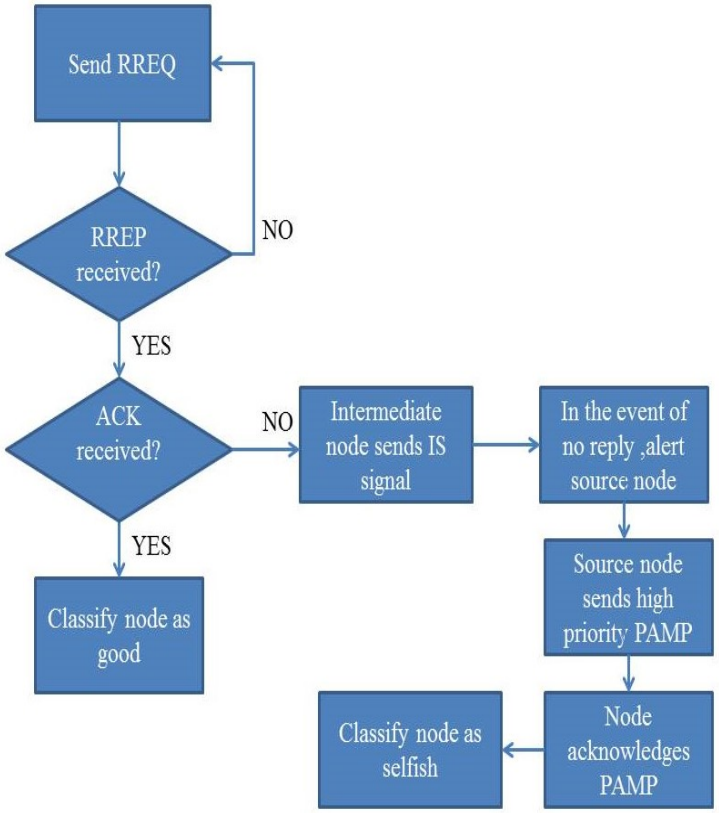

Figure 7-2 Proposed PrAIS algorithm flowchart

To identify the presence of a selfish node, the PAMP signal is sent. If the node is selfish it is forced to acknowledge receipt of PAMP. On the other hand, if the node is not able to send a Route Reply or acknowledgment, DS is sent to identify the route error, since due to route error the node will not be able to send a reply nor acknowledgment and therefore the preceding nodes are informed of the route error. The flowchart of the proposed PrAIS can be seen in Figure. 7-2. The pain equations P_{ba} and P_{aa} can be modelled as

$$P_{ba} = E_{cost}\left(w_{ss1}.T_{x,y}^{SC1} + w_{ss2}.T_{x,y}^{SC2}\right) + Infl(t) \quad (7\text{-}9)$$

Where Trust Purpose =SC1, SC2 in P_{ba}

$$P_{aa} = E_{cost}\left(w_{PAMP}.T_{x,y}^{PAMP} + w_{IS}.T_{x,y}^{IS}\right) + Infl(t) \quad (7\text{-}10)$$

Where Trust Purpose =IS, PAMP in P_{aa}. The Pain Reduction Equation (P_{RE}) (7-11) forms the crux of the PrAIS algorithm as it gives the change in pain after sending the PAMP and DS signals so that the presence of selfish node or route error can be identified and validated.

$$P_{RE=}|P_{ba} - P_{aa}| \quad (7\text{-}11)$$

7.4 Simulation and Results

The ns-3.23 simulator was used to detect and confirm the presence of a PST attacker using the AODV protocol. Extensive simulations were carried out using NS-3.23 to verify the mathematical formulation presented. Pause time is used to describe the interval between the mobility of nodes.By having a pause time of infiniteness implies that the node is stationary and by having a pause time of zero implies that the node is highly mobile.Therefore with the help of pause time it is possible to achieve a dynamicity of either high/medium or fixed nodes.The simulation parameters used are shown in Table 7-1. Selfish nodes were also introduced in the simulation.

As can be seen in Figure 7-3, as the trust reduces in the network, the Inflammation on the node increases. As delays increase, node mobility decreases which causes uncertainty for any two nodes to come in direct contact and create trust. Such a scenario that reduces mobility causes decreasing trust and increasing inflammation. The network is subjected to selfish nodes and in AODV, the packet delivery ratio will reduce as there is no antidote to curb selfishness, whereas in PrAIS the presence of IS and PAMP alleviates pain thereby achieving an improved packet delivery ratio, Figure.7-4.

The average end-to-end delay is the average time between data packets sent out from the

source node and received at the destination node. The end-to-end delay decreases for PrAIS as shown in Figure 7-.5 as the selfish nodes are avoided due to PAMP signals whereas the routing path taken by AODV will have selfish nodes thereby increasing the packet delivery delay.

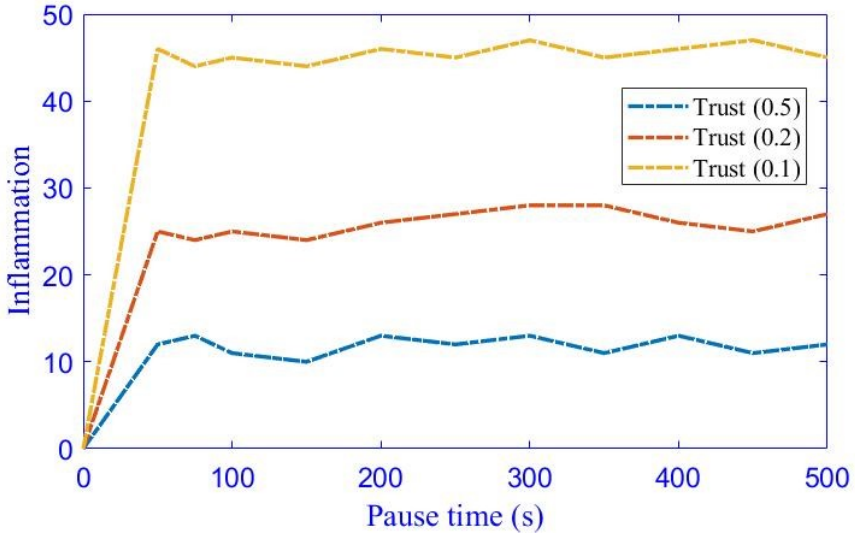

Figure 7-3 Inflammation v/s Pause time

As can be seen in Figure 7-6, the routing overhead is the ratio of route discovery packets to the data packets sent by source. The routing overhead for PrAIS is lower as it chooses a route avoiding selfish nodes resulting in less need to reconstruct routes during data transfer whereas in AODV the need to initiate route discovery is higher, as the effect of selfish nodes cannot be alleviated.

Table 7-1 Simulated Parameters

Simulator	Ns-3.23
Mobility Model	Random waypoint
Simulation Time	900s
Number of nodes	10-50
Traffic Type	UDP
Network Area	300m*1500m

Mobility	20 m/s
Pause Time	0-500 s
Transmission Range	50m

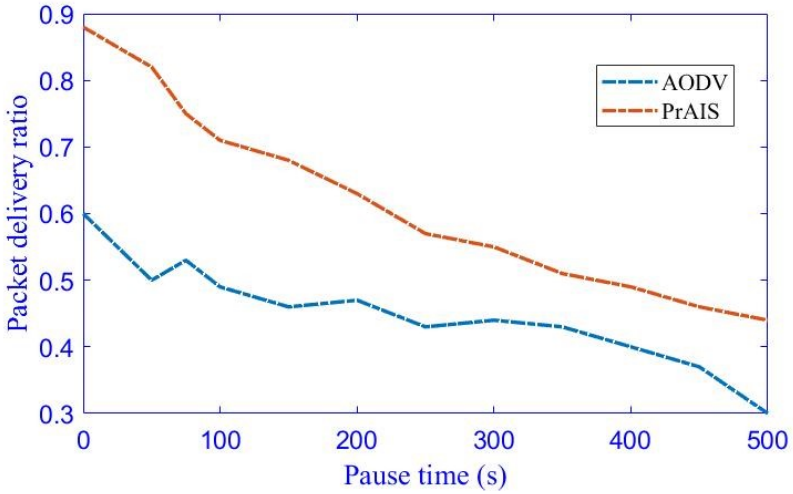

Figure 7-4 Packet delivery ratio v/s Pause time

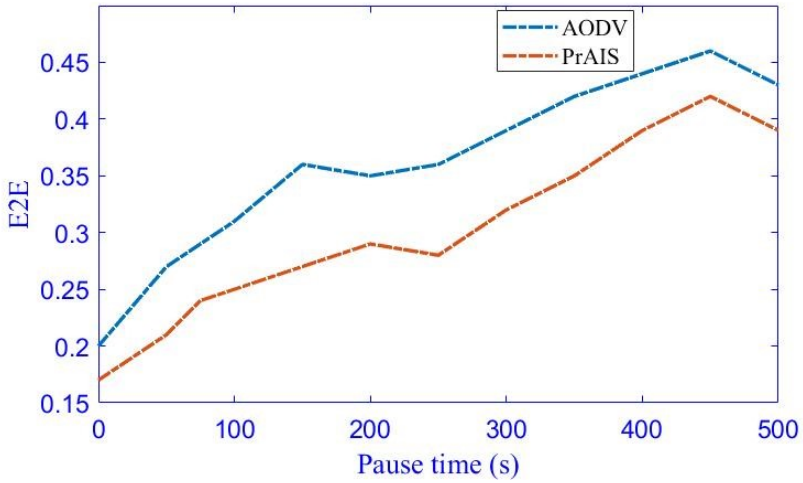

Figure 7-5 E2E v/s Pause time

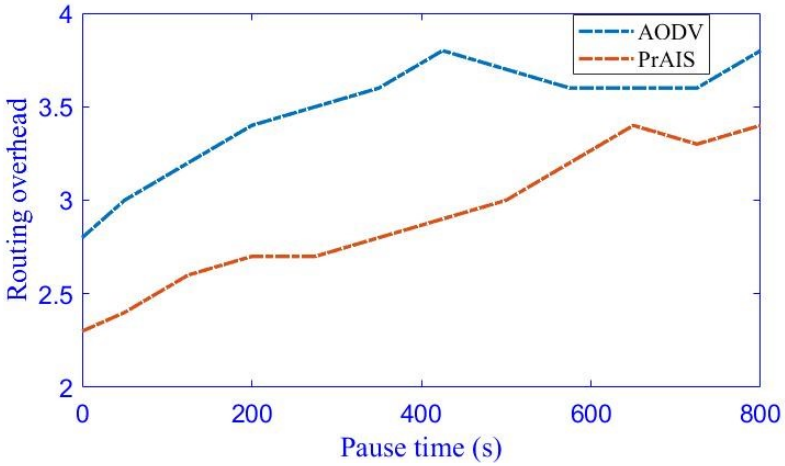

Figure 7-6 Routing overhead v/s Pause time

7.5 Summary

In this chapter, the research introduced a new approach to dealing with MANET pain, which is a HIS concept applied to MANET using AIS techniques. The factors used to identify pain in MANET were Energy and Trust. MANET pain was alleviated with use of the novel PrAIS algorithm.

Chapter 8 CONCLUSION AND FUTURE WORK

The research successfully answered the research questions and the results presented in this thesis make a significant contribution to the body of knowledge in the application of AIS to improve MANET security and performance. A new and novel algorithm has been presented that improves MANET security whilst a variant improves performance by reducing the effect of selfish nodes.

A review of the literature and a thematic classification of various AIS algorithms are provided in Chapter 2. A classification is proposed according to the challenges that AIS based MANET schemes might attempt to solve, thus providing a more efficient understanding of the proposed solution. In addition, the security attacks in MANET have also been detailed thereby providing an understanding of the reason behind the investigation of a new routing attack in MANET.

In the current state of the art the nodes are in a protected state or human intervention is required when facing security threats. These situations are impractical in a MANET which is known for dynamic topology and node mobility. Therefore, a new and innovative approach is necessary which can overcome the challenges of the existing MANET design and rectify the drawbacks of the current state of the art. Integration of an AIS scheme in MANET packet transmission to create AIS based routing (Translate AIS signals to MANET signals) has the potential to be a valuable framework.

A model of an AIS based security algorithm was developed where each node is modelled as a DC that initiates immune responses. Each DC node monitors the routing process and generates signals indicating the presence or absence of danger.

AISBA was designed with AIS signals to provide a secure routing algorithm to detect selfish nodes. This was inspired from the HIS as the DCs in the Human body provide a robust defence. To guarantee reliability and minimizing end-to-end latency, Trust metrics have been modelled and utilized to provide secure routing for MANET nodes. Extensive simulations demonstrate that AISBA yields a significant improvement in detection rate and packet delivery ratio.

A novel routing attack, PST, was conceptualised and modelled in a MANET. In PST, the attacker modifies the storage time of the packet so that it does not reach the intended destination nodes. Utilizing AIS signals the source of the PST attack was successfully identified. The potential for PST to be detrimental to MANET is considerable and the solution to this attack has been presented with an analysis that provides evidence that the operation of MANETs is susceptible to malicious attack without an improved security regime.

The GrAIS was developed in response to the loss of performance found when selfish nodes fail to participate in MANET opertion. A variant of the AISBA, GrAIS, takes advantage of the idea of a Dawkins model of birds and transforms the issue of selfish nodes non-cooperation by stimulating them to cooperate by utilizing the concept of increasing workload. Simulation results show that GrAIS yields significant improvements in the efficiency of packet delivery.

The concept of pain was introduced and modelled for a MANET. The PrAIS is a new and innovative approach to identifying and dealing with MANET pain, which is analogous to

what is found in the HIS. PrAIS applies a Pain before action (P_{ba}) and Pain after action (P_{aa}) based Pain Reduction approach, which uses the AIS signals, and trust among the nodes. Extensive simulations have demonstrated the efficiency and effectiveness of the proposed approach.

There are considerable opportunities for future work in the area of applying HIS concepts to AIS applied solutions to evolving issues with MANET. In MANET pain could, for example, be broadened to incorporate other parameters apart from trust and energy. The AISBA framework led to the formation of GrAIS and PrAIS algorithm. Similarly, the AISBA framerwork could be broadened to pave the way for other AIS algorithms that focus on improving MANET security.

BIBLIOGRAPHY

[1] F. Gu, J. Greensmith, and U. Aickelin, "The dendritic cell algorithm for intrusion detection," Bio-Inspired Communications and Networking, IGI Global, pp. 84-102, 2011.

[2] Loo J.Mauri JL, Ortiz JH. Mobile Ad hoc networks: current status and future trend. 1st Edn, CRC Press Publisher.,ISBN-10: 1439856508, pp: 538, 2012.

[3] Giordano, S. "Mobile ad hoc networks." Handbook of wireless networks and mobile computing: 325-346.2002,

[4] Jawandhiya Pradip M ,Ghonge, Mangesh ,M Ali, MS Deshpande, "A survey of mobile ad hoc network attacks," International Journal of Engineering Science and Technology 2(9): 4063-4071, 2010.

[5] Adjih, C., et al. "Attacks against OLSR: Distributed key management for security". 2_{nd} OLSR Interop and Workshop.2005.

[6] Goyal, P., et al. "Manet: vulnerabilities, challenges, attacks, application." IJCEM International Journal of Computational Engineering & Management 11(2011): pp 32-37.2011.

[7] Bala, A., et al. "Performance analysis of MANET under blackhole attack." First International Conference on Networks and Communications, NETCOM'09, IEEE.2009, pp. 141-145.

[8] Mistry, N., et al. "Improving AODV protocol against blackhole attacks." international multiconference of engineers and computer scientists.2010,Vol.2, March 2010.

[9] Bandyopadhyay, A., et al. "A simulation analysis of flooding attack in MANET using NS-3." Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology (Wireless VITAE), 2nd International Conference on, IEEE.2011.

[10] Yi, P., et al. "A new routing attack in mobile ad hoc networks." International Journal of Information Technology ,vol.11no.2: pp 83-94.2005.

[11] Mahajan, V., et al. "Analysis of wormhole intrusion attacks in MANETS". Military Communications Conference . MILCOM . IEEE, November 2008,pp. 1-7.

[12] Abdelhaq, M., et al. Using dendritic cell algorithm to detect the resource consumption attack over MANET. 181 CCIS: pp.429-442.2011.

[13] Karlsson, L. S. Dooley, and G. Pulkkis, "A new MANET wormhole detection algorithm based on traversal time and hop count analysis," Sensors, vol. 11, pp. 11122-11140, 2011.

[14] R. Maulik and N. Chaki, "A study on wormhole attacks in MANET," International Journal of Computer Information Systems and Industrial Management Applications ISSN, pp. 2150-7988, 2011.

[15] N. Meghanathan, "Stability and hop count of node-disjoint and link-disjoint multi-path routes in ad hoc networks," in Proceedings of the 3rd International IEEE Conference on Wireless and Mobile Computing, Networking and Communications, 2007. WiMOB 2007. pp. 42-42.

[16] K. Sridhar and M. C. Chan, "Stability and hop-count based approach for route computation in MANET," in Computer Communications and Networks, 2005. ICCCN 2005. Proceedings. 14th International Conference on, 2005, pp. 25-31.

[17] A. Lindgren, A. Doria, and O. Schele´n, "Probabilistic Routing in Intermittently Connected Networks," Proc. First Int'l Workshop Service Assurance with Partial and Intermittent Resources (SAPIR '04), pp. 239-254, Aug. 2004.

[18] S.-M. Jen, C.-S. Laih, and W.-C. Kuo, "A hop-count analysis scheme for avoiding wormhole attacks in MANET," Sensors, vol. 9, pp. 5022-5039, 2009.

[19] W. Hsu and A. Helmy, "On Nodal Encounter Patterns in Wireless LAN Traces," Proc. Fourth Int'l Symp. Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt '06), pp. 1-10, Apr.2006.

[20] W.-J. Hsu and A. Helmy, "Impact: Investigation of Mobile-User Patterns across University Campuses Using WLAN Trace Analysis," Proc. IEEE INFOCOM, Aug. 2005.

[21] E. M. Daly and M. Haahr, "Social network analysis for information flow in disconnected delay-tolerant MANETs," Mobile Computing, IEEE Transactions on, vol. 8, pp. 606-621, 2009.

[22] J. Burgess, B. Gallagher, D. Jensen, and B.N. Levine, "Maxprop: Routing for Vehicle-Based Disruption-Tolerant Networking,"Proc. IEEE INFOCOM, Mar. 2006.

[23] K. Tan, Q. Zhang, and W. Zhu, "Shortest Path Routing in Partially Connected Ad Hoc Networks," Proc. IEEE Global Telecomm. Conf.(GLOBECOM '03), vol. 2, pp. 1038-1042, Dec. 2003.

[24] A. Khelil, P.J. Marron, and K. Rothermel, "Contact-Based MobilityMetrics for Delay-Tolerant Ad Hoc Networking," Proc. 13th IEEE Int'l Symp. Modeling, Analysis, and Simulation of Computer and Telecomm. Systems (MASCOTS '05) pp. 435-444, 2005.

[25] M. Grossglauser and M. Vetterli, "Locating Nodes with Ease: Last Encounter Routing in Ad Hoc Networks through Mobility Diffusion," Proc. IEEE INFOCOM, vol. 3, pp. 1954-1964, 2003.

[26] J. Ghosh, H.Q. Ngo, and C. Qiao, “Mobility Profile Based Routing within Intermittently Connected Mobile Ad Hoc Networks (ICMAN),” Proc. Int’l Conf. Wireless,pp. 551-556, 2006.

[27] J. Lebrun, C.-N. Chuah, D. Ghosal, and M. Zhang, “Knowledge-Based Opportunistic Forwarding in Vehicular Wireless Ad Hoc Networks,” Proc. IEEE 61st Conf. Vehicular Technology (VTC-Spring’05), vol. 4, pp. 2289-2293, May 2005

[28] J. Leguay, T. Friedman, and V. Conan, “Evaluating Mobility Pattern Space Routing for DTNs,” Proc. IEEE INFOCOM, Apr. 2006.

[29] V. Mahajan, M. Natu, and A. Sethi, "Analysis of wormhole intrusion attacks in MANETS," in Military Communications Conference, 2008. MILCOM 2008. IEEE, 2008, pp. 1-7.

[30] S. Choi, D. Kim, D. Lee, J. Jung. “WAP: Wormhole Attack Prevention Algorithm in Mobile Ad Hoc Networks”. In International Conference on Sensor Networks, Ubiquitous and Trustworthy Computing, pp. 343-348, 2008.

[31] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (AODV) routing," 2070-1721, 2003.

[32] D. Vivian, E.A.P. Alchieri, C.B. Westphall. “Evaluation of QoS Metrics in Ad Hoc Networks with the use of Secure Routing Protocols”. In Network Operations and Management Symposium, pp. 1-14, 2006.

[33] T. Clausen and P. Jacquet, "Optimized link state routing protocol (OLSR)," 2070-1721, 2003.

[34] E. M. Royer and C.-K. Toh, "A review of current routing protocols for ad hoc mobile wireless networks," IEEE personal communications, vol. 6, pp. 46-55, 1999.

[35] S. Marti, T. J. Giuli, K. Lai, and M. Baker, "Mitigating routing misbehavior in mobile ad hoc networks," in Proceedings of the 6th annual international conference on Mobile computing and networking, 2000, pp. 255-265.

[36] Y. Yoo and D. P. Agrawal, "Why does it pay to be selfish in a MANET?," IEEE Wireless Communications, vol. 13, pp. 87-97, 2006.

[37] D. G. Kampitaki, E. D. Karapistoli, and A. A. Economides, "Evaluating selfishness impact on MANETs," in Telecommunications and Multimedia (TEMU), 2014 International Conference on, 2014, pp. 64-68.

[38] S. Buchegger and J.Y. Le Boudec, "Nodes bearing grudges: Towards routing security, fairness, and robustness in mobile ad hoc networks," in Parallel, Distributed and Network-based Processing, 2002. Proceedings. 10th Euromicro Workshop on, 2002, pp. 403-410.

[39] S. Gupta, C. Nagpal, and C. Singla, "Impact of selfish node concentration in MANETs," International Journal of Wireless & Mobile Networks (IJWMN) Vol, vol. 3, pp. 29-37, 2011.

[40] M. T. Tran and V. Simon, "Can altruism spare energy in ad hoc networking?" in Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia, 2011, pp. 214-217.

[41] Maha Abdelhaq, Rosilah Hassan, Mahamod Ismail, Raed Alsaqour, Daud Israf "Detecting Sleep Deprivation Attack over MANET Using a Danger Theory –Based Algorithm" International Journal on New Computer Architectures and Their Applications (IJNCAA) 1(3): pp. 534-541 The Society of Digital Information and Wireless Communications, 2011 (ISSN: 2220-9085)

[42] Julie Greensmith University of Nottingham "The Dendritic Cell Algorithm" Thesis submitted for the degree of Doctor of Philosophy

[43] S. Forrest, A.S. Perelson, L. Allen, R. Cherukuri, Self–nonself discrimination in a computer, IEEE Symposium on Research in Security and Privacy, Los Alamitos,CA, 1994.

[44] J.D. Farmer, N.H. Packard, A.S. Perelson, The immune system, adaptation, and machine learning, Physica D 22 (1986),pp. 187–204.

[45] Y. Ishida, Fully distributed diagnosis by PDP learning algorithm: towards immune network PDP model, IEEE International Joint Conference on Neural Networks, San Diego, USA, 1990.

[46] J. Timmis, M. Neal, J. Hunt, An artificial immune system for data analysis, Biosystems 55 (2000) ,pp. 143–150

[47] L.N.D. Castro, F.J.V. Zuben, The clonal selection algorithm with engineering applications, Genetic and Evolutionary Computation Conference (GECCO'00)– Workshop Proceedings, Las Vegas, Nevada, USA, 2000.

[48] L.N.D. Castro, J. Timmis, Artificial Immune Systems: A New Computational Intelligence Approach, Springer-Verlag, London, 2002.

[49] A.O. Tarakanov, V.A. Skormin, S.P. Sokolova, Immunocomputing: Principles and Applications, Springer, New York, 2003.

[50] Y. Ishida, Immunity-based Systems: A Design Perspective,Springer, 2004.

[51] D. Dasgupta, An overview of artificial immune systems in: D.Dasgupta (Ed.),Artificial Immune Systems and Their Applications, Springer-Verlag, 1998, pp. 3–19.

[52] G. Levy, Where numerics matter: matter: an introduction to quasi-random numbers, Financial Engineering News (2002).

[53] X.Z. Gao, S.J. Ovaska, X. Wang, Genetic algorithms-based detector generation in negative selection algorithm, in: 2006 IEEE Mountain Workshop on Adaptive and Learning Systems, 2006.

[54] W. Ma, D. Tran, D. Sharma, Negative selection with antigen feedback in intrusion detection, in: 7th international conference on Artificial Immune Systems, Phuket, Thailand, 2008.

[55] B. Caldas, M. Pita, F. Buarque, How to obtain appropriate executive decisions using artificial immunologic systems, in: 6th International Conference on Artificial Immune Systems (ICARIS 2007), Santos, Brazil, 2007.

[56] A.J. Graaff, A.P. Engelbrecht, Optimized coverage of non-self with evolved lymphocytes in an artificial immune system International Journal of Computational Intelligence Research (IJCIR) 2 (2006) ,pp. 127–150.

[57] J.L.M. Amaral, J.F.M. Amaral, R. Tanscheit, Real-valued negative selection algorithm with a Quasi-Monte Carlo genetic detector generation, in: 6th International Conference on Artificial Immune Systems (ICARIS 2007), Santos, Brazil, 2007.

[58] U. Aickelin, S. Cayzer, The danger theory and its application to artificial immune systems, in: The 1st International Conference on Artificial Immune Systems (ICARIS 2002), Canterbury, England, 2002.

[59] J Greensmith, U. Aickelin, J. Twycross, Detecting danger: applying a novel immunological concept to intrusion detection systems, in: 6th International Conference in Adaptive Computing in Design and Manufacture (ACDM 2004 Poster), Bristol, UK, 2004.

[60] N.K. Jerne, Towards a network theory of the immune system, Annals of Immunology (Paris) 125C (1974),pp.373–389.

[61] J.E. Hunt, D.E. Cooke, Learning using an artificial immune system, Journal of Network and Computer Applications 19 (1996),pp. 189–212.

[62] J. Pacheco, J.F. Costa, The abstract immune system algorithm, in: 6th International Conference on Unconventional Computation, Kingston, Canada, 2007.

[63] G.P. Coelho, F.J.V. Zuben, Omni-aiNet: an immune-inspired approach for omnioptimization, in: 5th International Conference on Artificial Immune Systems (ICARIS 2006), Oeiras, Portugal, 2006.

[64] G.P. Coelho,F.O.d. Franca, F.J.V. Zuben,A multi-objective multipopulation approach for biclustering in:7th International Conference on Artificial Immune Systems,Phuket,Thaileand 2008.

[65] T. Stibor, J. Timmis, An investigation on the compression quality of aiNet, in:IEEE Symposium on Foundations of Computational Intelligence (FOCI 2007),2007.

[66] V. Cutello, G. Narzisi, G. Nicosia, and M. Pavone, "Clonal selection algorithms: a comparative case study using effective mutation potentials," in International Conference on Artificial Immune Systems, 2005, pp. 13-28.

[67] L.N.D. Castro, F.J.V. Zuben, Learning and optimization using the clonal selection principle, IEEE Transactions on Evolutionary Computation, vol. 6, 2002, pp. 239–251.

[68] A. Ciccazzo, P. Conca, G. Nicosia, and G. Stracquadanio, "An advanced clonal selection algorithm with ad-hoc network-based hypermutation operators for synthesis of topology and sizing of analog electrical circuits," in International Conference on Artificial Immune Systems, 2008, pp. 60-70.

[69] R. Halavati, S.B. Shouraki, M.J. Heravi, B.J. Jashmi, An artificial immune system with partially specified antibodies, in: 9th annual conference on Genetic and evolutionary computation (GECCO 2007), London, England, 2007.

[70] P. May, J. Timmis, K. Mander, Immune and evolutionary approaches to software mutation testing, in: 6th International Conference on Artificial Immune Systems (ICARIS 2007), Santos, Brazil, 2007.

[71] W.O. Wilson, P. Birkin, U. Aickelin, Price trackers inspired by immune memory, in: 5th International Conference on Artificial Immune Systems (ICARIS 2006), Oeiras, Portugal, 2006.

[72] P. Matzinger, The danger model: a renewed sense of self, Science 296 (2002),pp. 301–305.

[73] P. Bretscher, M. Cohn, A theory of self-non self discrimination, Science 169(1970),pp.1042–1049.

[74] U. Aickelin and S. Cayzer, "The danger theory and its application to artificial immune systems," Browser Download This Paper, 2002.

[75] J. Greensmith, U. Aickelin, and J. Twycross, "Detecting danger: Applying a novel immunological concept to intrusion detection systems," 2004.

[76] U. Aickelin and S. Cayzer, "The danger theory and its application to artificial immune systems," Browser Download This Paper, 2002.

[77] C. E. Prieto, F. Nino, and G. Quintana, "A goalkeeper strategy in robot soccer based on Danger Theory," in Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on, 2008, pp. 3443-3447.

[78] A. Secker, A. Freitas, and J. Timmis, "Towards a danger theory inspired artificial immune system for web mining," Web Mining: Applications and Techniques, Idea Group, pp. 145-168, 2005.

[79] R. M. Steinman and Z. A. Cohn, "Identification of a novel cell type in peripheral lymphoid organs of mice," Journal of Experimental Medicine, vol. 137, pp. 1142-1162, 1973.

[80] M. L. Kapsenberg, "Dendritic-cell control of pathogen-driven T-cell polarization," Nature Reviews Immunology, vol. 3, pp. 984-993, 2003.

[81] T. Jamie and U. Aickelin, "Towards a conceptual framework for innate immunity," in 3rd International Conference on Artificial Immune Systems (ICARIS 2004), Catania, Italy, 2004.

[82] J. Greensmith and U. Aickelin, "The deterministic dendritic cell algorithm," in International Conference on Artificial Immune Systems, 2008, pp. 291-302.

[83] J. Greensmith, U. Aickelin, and J. Twycross, "Articulation and clarification of the dendritic cell algorithm," in International Conference on Artificial Immune Systems, 2006, pp. 404-417.

[84] F. Gu, J. Greensmith, and U. Aickelin, "Further exploration of the dendritic cell algorithm: Antigen multiplier and time windows," in International Conference on Artificial Immune Systems, 2008, pp. 142-153.

[85] H. G. Kayacik, A. N. Zincir-Heywood, and M. I. Heywood, "Selecting features for intrusion detection: A feature relevance analysis on KDD 99 intrusion detection datasets," in Proceedings of the third annual conference on privacy, security and trust, 2005.

[86] R. Oates, J. Greensmith, U. Aickelin, J. Garibaldi, and G. Kendall, "The application of a dendritic cell algorithm to a robotic classifier," in Artificial Immune Systems, ed: Springer, 2007, pp. 204-215.

[87] http://users.isy.liu.se/en/rt/andrecb/fidodido/doc/AriaReference.pdf

[88] J. Kim, P. Bentley, C. Wallenta, M. Ahmed, and S. Hailes, "Danger is ubiquitous: Detecting malicious activities in sensor networks using the dendritic cell algorithm," in International Conference on Artificial Immune Systems, 2006, pp. 390-403.

[89] A. O. Bang and P. L. Ramteke, "Manet: history, challenges and applications," International Journal of Application or Innovation in Engineering & Management (IJAIEM), vol. 2, pp. 249-251, 2013.

[90] R. Akbani, Defending against malicious nodes in closed MANETs through packet authentication and a hybrid trust management system: The University of Texas at San Antonio, 2009.

[91] Y. A. Mohamed and A. Abdullah, "I2MANET Security Logical Specification Framework," Int. Arab J. Inf. Technol., vol. 9, pp. 495-503, 2012.

[92] F. Kargl, A. Geis, S. Schlott, and M. Weber, "Secure dynamic source routing," in System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on, 2005, pp. 320c-320c.

[93] M. Abdelhaq, R. Hassan, M. Ismail, and D. Israf, "Detecting resource consumption attack over MANET using an artificial immune algorithm," Research Journal of Applied Sciences, Engineering and Technology, vol. 3, pp. 1026-1033, 2011.

[94] S. Kumari and M. Shrivastava, "Secure DSR protocol in MANET using energy efficient intrusion detection system," International Journal, vol. 1, 2012.

[95] P. Goyal, V. Parmar, and R. Rishi, "Manet: vulnerabilities, challenges, attacks, application," IJCEM International Journal of Computational Engineering & Management, vol. 11, pp. 32-37, 2011.

[96] S. Sarafijanović and J.-Y. Le Boudec, "An artificial immune system for misbehavior detection in mobile ad-hoc networks with virtual thymus, clustering, danger signal, and memory detectors," in International Conference on Artificial Immune Systems, 2004, pp. 342-356.

[97] S. A. Hofmeyr and S. Forrest, "Architecture for an artificial immune system," Evolutionary computation, vol. 8, pp. 443-473, 2000.

[98] Y. A. Mohamed and A. B. Abdullah, "Implementation of IDS with response for securing MANETs," in Information Technology (ITSim), 2010 International Symposium in, 2010, pp. 660-665.

[99] N. Mazhar and M. Farooq, "A sense of danger: dendritic cells inspired artificial immune system for manet security," in Proceedings of the 10th annual conference on Genetic and evolutionary computation, 2008, pp. 63-70.

[100] X. Ye and J. Li, "A security architecture based on immune agents for MANET," in Wireless Communication and Sensor Computing, 2010. ICWCSC 2010. International Conference on, 2010, pp. 1-5.

[101] F. Barani, "A hybrid approach for dynamic intrusion detection in ad hoc networks using genetic algorithm and artificial immune system," in Intelligent Systems (ICIS), 2014 Iranian Conference on, 2014, pp. 1-6.

[102] N. Mazhar, "Energy efficient security in MANETs: A comparison of cryptographic and artificial immune systems," Pakistan Journal of Engineering and Applied Sciences, 2016.

[103] M. S. A. Ansari and M. Inamullah, "Misbehavior detection in mobile ad hoc networks using Artificial Immune System approach," in Advanced Networks and Telecommunication Systems (ANTS), 2011 IEEE 5th International Conference on, 2011, pp. 1-6.

[104] S. Sarafijanovic and J.-Y. Le Boudec, "An artificial immune system approach with secondary response for misbehavior detection in mobile ad hoc networks," IEEE Transactions on Neural Networks, vol. 16, pp. 1076-1087, 2005.

[105] J. Kim and P. J. Bentley, "An evaluation of negative selection in an artificial immune system for network intrusion detection," in Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation, 2001, pp. 1330-1337.

[106] F. Barani and M. Abadi, "An ABC-AIS hybrid approach to dynamic anomaly detection in AODV-based MANETs," in Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on, 2011, pp. 714-720.

[107] A. Khannous, A. Rghioui, F. Elouaai, and M. Bouhorma, "Manet security: An intrusion detection system based on the combination of negative selection and danger theory concepts," in Next Generation Networks and Services (NGNS), 2014 Fifth International Conference on, 2014, pp. 88-91.

[108] A. Visconti and H. Tahayori, "Detecting misbehaving nodes in MANET with an artificial immune system based on type-2 fuzzy sets," in Internet Technology and Secured Transactions, 2009. ICITST 2009. International Conference for, 2009, pp. 1-2.

[109] J. M. Mendel, "Computing with words and its relationships with fuzzistics," Information Sciences, vol. 177, pp. 988-1006, 2007.

[110] Y. A. Mohamed and A. B. Abdullah, "Immune-inspired framework for securing hybrid MANET," in Industrial Electronics & Applications, 2009. ISIEA 2009. IEEE Symposium on, 2009, pp. 301-306.

[111] A. Khannous, A. Rghioui, F. Elouaai, and M. Bouhorma, "Securing MANETS using the integration of concepts from diverse immune theories ," Journal of Theoretical and Applied Information Technology, vol. 88, p. 35, 2016.

[112] K. D. Elgert, Immunology: understanding the immune system: John Wiley & Sons, 2009.

[113] R. Dawkins, The Selfish Gene. New York: Oxford, 2006.

[114] D. Cerri and A. Ghioni, "Securing AODV: the A-SAODV secure routing prototype," IEEE Communications Magazine, vol. 46, 2008.

[115] F. Ye et al., A scalable solution to minimum cost forwarding in large scale sensor networks, in: Proceedings of International Conference on Computer Communications and Networks (ICCCN), Dallas, TX, October 2001

[116] B. Yang, Y. Chen, X. Jiang, B. Yang, Y. Cai, and Y. Cai, "Packet delivery ratio/cost in MANETs with erasure coding and packet replication," IEEE Trans. Veh. Technol. IEEE Transactions on Vehicular Technology, vol. 64, pp. 2062-2070, 2015.

Appendix 1 aisba.cc

```
#include "ns3/aisba-module.h"
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/wifi-module.h"
#include "ns3/v4ping-helper.h"
#include <iostream>
#include <cmath>
// Lincy
#include "ns3/aisba-rtable.h"

using namespace ns3;
using namespace ns3::aisba;

*/
class AisbaExample
{
public:
  AisbaExample ();
  /// Configure script parameters, \return true on successful configuration
  bool Configure (int argc, char **argv);
  /// Run simulation
  void Run ();
  /// Report results
  void Report (std::ostream & os);
  /// Lincy
  //void activateDC();
private:

  // parameters
  /// Number of nodes
  uint32_t size;
  /// Distance between nodes, meters
  double step;
  /// Simulation time, seconds
  double totalTime;
  /// Write per-device PCAP traces if true
  bool pcap;
  /// Print routes if true
  bool printRoutes;

  // network
  NodeContainer nodes;
  NetDeviceContainer devices;
  Ipv4InterfaceContainer interfaces;
```

```
private:
  void CreateNodes ();
  void CreateDevices ();
  void InstallInternetStack ();
  void InstallApplications ();
};

int main (int argc, char **argv)
{
  AisbaExample test;
  if (!test.Configure (argc, argv))
    NS_FATAL_ERROR ("Configuration failed. Aborted.");

  test.Run ();
  test.Report (std::cout);
  return 0;
}

//-----------------------------------------------------------------------------
AisbaExample::AisbaExample () :
  size (200),
  step (70),
  totalTime (10),
  pcap (true),
  printRoutes (true)
{
}

bool
AisbaExample::Configure (int argc, char **argv)
{
  //
  // LogComponentEnable("AisbaRoutingProtocol", LOG_LEVEL_ALL);

  SeedManager::SetSeed (12345);
  CommandLine cmd;

  cmd.AddValue ("pcap", "Write PCAP traces.", pcap);
  cmd.AddValue ("printRoutes", "Print routing table dumps.", printRoutes);
  cmd.AddValue ("size", "Number of nodes.", size);
  cmd.AddValue ("time", "Simulation time, s.", totalTime);
  cmd.AddValue ("step", "Grid step, m", step);

  cmd.Parse (argc, argv);
  return true;
}

void
AisbaExample::Run ()
```

```
{
//  Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold", UintegerValue (1)); // enable
rts cts all the time.
  CreateNodes ();
  CreateDevices ();
  InstallInternetStack ();
  InstallApplications ();

  std::cout << "Starting simulation for " << totalTime << " s ...\n";

  Simulator::Stop (Seconds (totalTime));
  Simulator::Run ();
  Simulator::Destroy ();
}

void
AisbaExample::Report (std::ostream &)
{
}

void
AisbaExample::CreateNodes ()
{
  std::cout << "Creating " << (unsigned)size << " nodes " << step << " m apart.\n";
  nodes.Create (size);
  // Name nodes
  for (uint32_t i = 0; i < size; ++i)
    {
      std::ostringstream os;
      os << "node-" << i;
      Names::Add (os.str (), nodes.Get (i));
    }
  // Create static grid
  MobilityHelper mobility;
  mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                                 "MinX", DoubleValue (0.0),
                                 "MinY", DoubleValue (0.0),
                                 "DeltaX", DoubleValue (step),
                                 "DeltaY", DoubleValue (0),
                                 "GridWidth", UintegerValue (size),
                                 "LayoutType", StringValue ("RowFirst"));
  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  mobility.Install (nodes);
}

void
AisbaExample::CreateDevices ()
{
  NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();
  wifiMac.SetType ("ns3::AdhocWifiMac");
```

```
  YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
  YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
  wifiPhy.SetChannel (wifiChannel.Create ());
  WifiHelper wifi = WifiHelper::Default ();
  wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager", "DataMode", StringValue
("OfdmRate6Mbps"), "RtsCtsThreshold", UintegerValue (0));
  devices = wifi.Install (wifiPhy, wifiMac, nodes);

  if (pcap)
    {
      wifiPhy.EnablePcapAll (std::string ("aisba"));
    }
}

void
AisbaExample::InstallInternetStack ()
{
  AisbaHelper aisba;
  // you can configure AODV attributes here using aisba.Set(name, value)
  InternetStackHelper stack;
  stack.SetRoutingHelper (aisba); // has effect on the next Install ()
  stack.Install (nodes);
  Ipv4AddressHelper address;
  address.SetBase ("10.0.0.0", "255.0.0.0");
  interfaces = address.Assign (devices);

  if (printRoutes)
    {
      Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper> ("aisba.routes",
std::ios::out);
      aisba.PrintRoutingTableAllAt (Seconds (8), routingStream);
    }
}

void
AisbaExample::InstallApplications ()
{
  V4PingHelper ping (interfaces.GetAddress (size - 1));
  //V4PingHelper ping2(interfaces.GetAddress (size - 2));
  ping.SetAttribute ("Verbose", BooleanValue (true));
  //ping2.SetAttribute ("Verbose", BooleanValue (true));
  ApplicationContainer p = ping.Install (nodes.Get (0));
 // ApplicationContainer p2 = ping2.Install (nodes.Get (0));
  p.Start (Seconds (0));
  p.Stop (Seconds (totalTime) - Seconds (0.001));
  //p2.Start (Seconds (2));
 // p2.Stop (Seconds (totalTime) - Seconds (0.001));

  // move node away
  Ptr<Node> node = nodes.Get (size/2);
```

```
  Ptr<MobilityModel> mob = node->GetObject<MobilityModel> ();
  Simulator::Schedule (Seconds (totalTime/3), &MobilityModel::SetPosition, mob, Vector (1e5, 1e5,
1e5));
}

// Method added by Lincy

/*void AisbaExample::activateDC(){
  RoutingTableEntry _oRoutingTableEntry;
  if(_oRoutingTableEntry.GetFlag() ==INVALID)
  {
                std::cout<<"PAMP-DC .\n";
  }
  if(_oRoutingTableEntry.GetFlag() ==VALID)
  {
                std::cout << "DC - UP .\n";
  }
  if(_oRoutingTableEntry.GetFlag() ==IN_SEARCH)
  {
                std::cout << "PAMP - IN_SEARCH .\n";
  }
}*/
```

Appendix 2 aisba-routing-protocol.cc

```
#define NS_LOG_APPEND_CONTEXT                                   \
  if (m_ipv4) { std::clog << "[node " << m_ipv4->GetObject<Node> ()->GetId () << "] "; }

#include "aisba-routing-protocol.h"
#include "ns3/log.h"
#include "ns3/boolean.h"
#include "ns3/random-variable-stream.h"
#include "ns3/inet-socket-address.h"
#include "ns3/trace-source-accessor.h"
#include "ns3/udp-socket-factory.h"
#include "ns3/wifi-net-device.h"
#include "ns3/adhoc-wifi-mac.h"
#include "ns3/string.h"
#include "ns3/pointer.h"
#include <algorithm>
#include <limits>
#include "aisba-rqueue.h"//lincy

namespace ns3
{

NS_LOG_COMPONENT_DEFINE ("AisbaRoutingProtocol");

namespace aisba
{
NS_OBJECT_ENSURE_REGISTERED (RoutingProtocol);

/// UDP Port for AISBA control traffic
const uint32_t RoutingProtocol::AISBA_PORT = 654;

//------------------------------------------------------------------------------
/// Tag used by AISBA implementation

class DeferredRouteOutputTag : public Tag
{

public:
  DeferredRouteOutputTag (int32_t o = -1) : Tag (), m_oif (o) {}

  static TypeId GetTypeId ()
  {
    static TypeId tid = TypeId ("ns3::aisba::DeferredRouteOutputTag").SetParent<Tag> ()
      .SetParent<Tag> ()
      .SetGroupName("Aisba")
      .AddConstructor<DeferredRouteOutputTag> ()
    ;
```

```
  return tid;
}

TypeId  GetInstanceTypeId () const
{
  return GetTypeId ();
}

int32_t GetInterface() const
{
  return m_oif;
}

void SetInterface(int32_t oif)
{
  m_oif = oif;
}

uint32_t GetSerializedSize () const
{
  return sizeof(int32_t);
}

void  Serialize (TagBuffer i) const
{
  i.WriteU32 (m_oif);
}

void  Deserialize (TagBuffer i)
{
  m_oif = i.ReadU32 ();
}

void  Print (std::ostream &os) const
{
  os << "DeferredRouteOutputTag: output interface = " << m_oif;
}

private:
  /// Positive if output device is fixed in RouteOutput
  int32_t m_oif;
};

NS_OBJECT_ENSURE_REGISTERED (DeferredRouteOutputTag);


//-----------------------------------------------------------------------------
RoutingProtocol::RoutingProtocol () :
  RreqRetries (2),
  RreqRateLimit (10),
  RerrRateLimit (10),
  ActiveRouteTimeout (Seconds (3)),
  NetDiameter (35),
```

```
  NodeTraversalTime (MilliSeconds (40)),
  NetTraversalTime (Time ((2 * NetDiameter) * NodeTraversalTime)),
  PathDiscoveryTime ( Time (2 * NetTraversalTime)),
  MyRouteTimeout (Time (2 * std::max (PathDiscoveryTime, ActiveRouteTimeout))),
  HelloInterval (Seconds (1)),
  AllowedHelloLoss (2),
  DeletePeriod (Time (5 * std::max (ActiveRouteTimeout, HelloInterval))),
  NextHopWait (NodeTraversalTime + MilliSeconds (10)),
  BlackListTimeout (Time (RreqRetries * NetTraversalTime)),
  MaxQueueLen (64),
  MaxQueueTime (Seconds (30)),
  DestinationOnly (false),
  GratuitousReply (true),
  EnableHello (false),
  m_routingTable (DeletePeriod),
  m_queue (MaxQueueLen, MaxQueueTime),
  m_requestId (0),
  m_seqNo (0),
  m_rreqIdCache (PathDiscoveryTime),
  m_dpd (PathDiscoveryTime),
  m_nb (HelloInterval),
  m_rreqCount (0),
  m_rerrCount (0),
  m_htimer (Timer::CANCEL_ON_DESTROY),
  m_rreqRateLimitTimer (Timer::CANCEL_ON_DESTROY),
  m_rerrRateLimitTimer (Timer::CANCEL_ON_DESTROY),
  m_lastBcastTime (Seconds (0))
{
  m_nb.SetCallback (MakeCallback (&RoutingProtocol::SendRerrWhenBreaksLinkToNextHop, taisba));
}

TypeId
RoutingProtocol::GetTypeId (void)
{
  static TypeId tid = TypeId ("ns3::aisba::RoutingProtocol")
    .SetParent<Ipv4RoutingProtocol> ()
    .SetGroupName("Aisba")
    .AddConstructor<RoutingProtocol> ()
    .AddAttribute ("HelloInterval", "HELLO messages emission interval.",
                   TimeValue (Seconds (1)),
                   MakeTimeAccessor (&RoutingProtocol::HelloInterval),
                   MakeTimeChecker ())
    .AddAttribute ("RreqRetries", "Maximum number of retransmissions of RREQ to discover a route",
                   UintegerValue (2),
                   MakeUintegerAccessor (&RoutingProtocol::RreqRetries),
                   MakeUintegerChecker<uint32_t> ())
    .AddAttribute ("RreqRateLimit", "Maximum number of RREQ per second.",
                   UintegerValue (10),
                   MakeUintegerAccessor (&RoutingProtocol::RreqRateLimit),
                   MakeUintegerChecker<uint32_t> ())
    .AddAttribute ("RerrRateLimit", "Maximum number of RERR per second.",
                   UintegerValue (10),
                   MakeUintegerAccessor (&RoutingProtocol::RerrRateLimit),
```

```
                MakeUintegerChecker<uint32_t> ())
    .AddAttribute ("NodeTraversalTime", "Conservative estimate of the average one hop traversal time for
packets and should include "
                "queuing delays, interrupt processing times and transfer times.",
                TimeValue (MilliSeconds (40)),
                MakeTimeAccessor (&RoutingProtocol::NodeTraversalTime),
                MakeTimeChecker ())
    .AddAttribute ("NextHopWait", "Period of our waiting for the neighbour's RREP_ACK = 10 ms +
NodeTraversalTime",
                TimeValue (MilliSeconds (50)),
                MakeTimeAccessor (&RoutingProtocol::NextHopWait),
                MakeTimeChecker ())
    .AddAttribute ("ActiveRouteTimeout", "Period of time during which the route is considered to be valid",
                TimeValue (Seconds (3)),
                MakeTimeAccessor (&RoutingProtocol::ActiveRouteTimeout),
                MakeTimeChecker ())
    .AddAttribute ("MyRouteTimeout", "Value of lifetime field in RREP generating by taisba node = 2 *
max(ActiveRouteTimeout, PathDiscoveryTime)",
                TimeValue (Seconds (11.2)),
                MakeTimeAccessor (&RoutingProtocol::MyRouteTimeout),
                MakeTimeChecker ())
    .AddAttribute ("BlackListTimeout", "Time for which the node is put into the blacklist = RreqRetries *
NetTraversalTime",
                TimeValue (Seconds (5.6)),
                MakeTimeAccessor (&RoutingProtocol::BlackListTimeout),
                MakeTimeChecker ())
    .AddAttribute ("DeletePeriod", "DeletePeriod is intended to provide an upper bound on the time for which an
upstream node A "
                "can have a neighbor B as an active next hop for destination D, while B has invalidated the route to
D."
                " = 5 * max (HelloInterval, ActiveRouteTimeout)",
                TimeValue (Seconds (15)),
                MakeTimeAccessor (&RoutingProtocol::DeletePeriod),
                MakeTimeChecker ())
    .AddAttribute ("NetDiameter", "Net diameter measures the maximum possible number of hops between two
nodes in the network",
                UintegerValue (35),
                MakeUintegerAccessor (&RoutingProtocol::NetDiameter),
                MakeUintegerChecker<uint32_t> ())
    .AddAttribute ("NetTraversalTime", "Estimate of the average net traversal time = 2 * NodeTraversalTime *
NetDiameter",
                TimeValue (Seconds (2.8)),
                MakeTimeAccessor (&RoutingProtocol::NetTraversalTime),
                MakeTimeChecker ())
    .AddAttribute ("PathDiscoveryTime", "Estimate of maximum time needed to find route in network = 2 *
NetTraversalTime",
                TimeValue (Seconds (5.6)),
                MakeTimeAccessor (&RoutingProtocol::PathDiscoveryTime),
                MakeTimeChecker ())
    .AddAttribute ("MaxQueueLen", "Maximum number of packets that we allow a routing protocol to buffer.",
                UintegerValue (64),
                MakeUintegerAccessor (&RoutingProtocol::SetMaxQueueLen,
                                      &RoutingProtocol::GetMaxQueueLen),
```

```
                MakeUintegerChecker<uint32_t> ())
    .AddAttribute ("MaxQueueTime", "Maximum time packets can be queued (in seconds)",
                TimeValue (Seconds (30)),
                MakeTimeAccessor (&RoutingProtocol::SetMaxQueueTime,
                                  &RoutingProtocol::GetMaxQueueTime),
                MakeTimeChecker ())
    .AddAttribute ("AllowedHelloLoss", "Number of hello messages which may be loss for valid link.",
                UintegerValue (2),
                MakeUintegerAccessor (&RoutingProtocol::AllowedHelloLoss),
                MakeUintegerChecker<uint16_t> ())
    .AddAttribute ("GratuitousReply", "Indicates whether a gratuitous RREP should be unicast to the node
originated route discovery.",
                BooleanValue (true),
                MakeBooleanAccessor (&RoutingProtocol::SetGratuitousReplyFlag,
                                     &RoutingProtocol::GetGratuitousReplyFlag),
                MakeBooleanChecker ())
    .AddAttribute ("DestinationOnly", "Indicates only the destination may respond to taisba RREQ.",
                BooleanValue (false),
                MakeBooleanAccessor (&RoutingProtocol::SetDesinationOnlyFlag,
                                     &RoutingProtocol::GetDesinationOnlyFlag),
                MakeBooleanChecker ())
    .AddAttribute ("EnableHello", "Indicates whether a hello messages enable.",
                BooleanValue (true),
                MakeBooleanAccessor (&RoutingProtocol::SetHelloEnable,
                                     &RoutingProtocol::GetHelloEnable),
                MakeBooleanChecker ())
    .AddAttribute ("EnableBroadcast", "Indicates whether a broadcast data packets forwarding enable.",
                BooleanValue (true),
                MakeBooleanAccessor (&RoutingProtocol::SetBroadcastEnable,
                                     &RoutingProtocol::GetBroadcastEnable),
                MakeBooleanChecker ())
    .AddAttribute ("UniformRv",
                "Access to the underlying UniformRandomVariable",
                StringValue ("ns3::UniformRandomVariable"),
                MakePointerAccessor (&RoutingProtocol::m_uniformRandomVariable),
                MakePointerChecker<UniformRandomVariable> ())
  ;
  return tid;
}

void
RoutingProtocol::SetMaxQueueLen (uint32_t len)
{
  MaxQueueLen = len;
  m_queue.SetMaxQueueLen (len);
}
void
RoutingProtocol::SetMaxQueueTime (Time t)
{
  MaxQueueTime = t;
  m_queue.SetQueueTimeout (t);
}
```

```
RoutingProtocol::~RoutingProtocol ()
{
}

void
RoutingProtocol::DoDispose ()
{
  m_ipv4 = 0;
  for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::iterator iter =
         m_socketAddresses.begin (); iter != m_socketAddresses.end (); iter++)
    {
      iter->first->Close ();
    }
  m_socketAddresses.clear ();
  for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::iterator iter =
         m_socketSubnetBroadcastAddresses.begin (); iter != m_socketSubnetBroadcastAddresses.end (); iter++)
    {
      iter->first->Close ();
    }
  m_socketSubnetBroadcastAddresses.clear ();
  Ipv4RoutingProtocol::DoDispose ();
}

void
RoutingProtocol::PrintRoutingTable (Ptr<OutputStreamWrapper> stream) const
{
  *stream->GetStream () << "Node: " << m_ipv4->GetObject<Node> ()->GetId () << " Time: " <<
Simulator::Now ().GetSeconds () << "s ";
  m_routingTable.Print (stream);
}

int64_t
RoutingProtocol::AssignStreams (int64_t stream)
{
  NS_LOG_FUNCTION (taisba << stream);
  m_uniformRandomVariable->SetStream (stream);
  return 1;
}

void
RoutingProtocol::Start ()
{
  NS_LOG_FUNCTION (taisba);
  if (EnableHello)
    {
      m_nb.ScheduleTimer ();
    }
  m_rreqRateLimitTimer.SetFunction (&RoutingProtocol::RreqRateLimitTimerExpire,
                                    taisba);
  m_rreqRateLimitTimer.Schedule (Seconds (1));

  m_rerrRateLimitTimer.SetFunction (&RoutingProtocol::RerrRateLimitTimerExpire,
                                    taisba);
```

```
  m_rerrRateLimitTimer.Schedule (Seconds (1));

}

Ptr<Ipv4Route>
RoutingProtocol::RouteOutput (Ptr<Packet> p, const Ipv4Header &header,
                         Ptr<NetDevice> oif, Socket::SocketErrno &sockerr)
{
  NS_LOG_FUNCTION (taisba << header << (oif ? oif->GetIfIndex () : 0));
  if (!p)
    {
      NS_LOG_DEBUG("Packet is == 0");
      return LoopbackRoute (header, oif); // later
    }
  if (m_socketAddresses.empty ())
    {
      sockerr = Socket::ERROR_NOROUTETOHOST;
      NS_LOG_LOGIC ("No aisba interfaces");
      Ptr<Ipv4Route> route;
      return route;
    }
  sockerr = Socket::ERROR_NOTERROR;
  Ptr<Ipv4Route> route;
  Ipv4Address dst = header.GetDestination ();
  RoutingTableEntry rt;
  if (m_routingTable.LookupValidRoute (dst, rt))
    {
      route = rt.GetRoute ();
      NS_ASSERT (route != 0);
      NS_LOG_DEBUG ("Exist route to " << route->GetDestination () << " from interface " << route-
>GetSource ());
      if (oif != 0 && route->GetOutputDevice () != oif)
        {
          NS_LOG_DEBUG ("Output device doesn't match. Dropped.");
          sockerr = Socket::ERROR_NOROUTETOHOST;
          return Ptr<Ipv4Route> ();
        }
      UpdateRouteLifeTime (dst, ActiveRouteTimeout);
      UpdateRouteLifeTime (route->GetGateway (), ActiveRouteTimeout);
      return route;
    }

  // Valid route not found, in taisba case we return loopback.
  // Actual route request will be deferred until packet will be fully formed,
  // routed to loopback, received from loopback and passed to RouteInput (see below)
  uint32_t iif = (oif ? m_ipv4->GetInterfaceForDevice (oif) : -1);
  DeferredRouteOutputTag tag (iif);
  NS_LOG_DEBUG ("Valid Route not found");
  if (!p->PeekPacketTag (tag))
    {
      p->AddPacketTag (tag);
    }
  return LoopbackRoute (header, oif);
```

```
}

void
RoutingProtocol::DeferredRouteOutput (Ptr<const Packet> p, const Ipv4Header & header,
                                      UnicastForwardCallback ucb, ErrorCallback ecb)
{
  NS_LOG_FUNCTION (taisba << p << header);
  NS_ASSERT (p != 0 && p != Ptr<Packet> ());

  QueueEntry newEntry (p, header, ucb, ecb);
  bool result = m_queue.Enqueue (newEntry);
  if (result)
    {
      NS_LOG_LOGIC ("Add packet " << p->GetUid () << " to queue. Protocol " << (uint16_t)
header.GetProtocol ());
      RoutingTableEntry rt;
      bool result = m_routingTable.LookupRoute (header.GetDestination (), rt);
      if(!result || ((rt.GetFlag () != IN_SEARCH) && result))
        {
          NS_LOG_LOGIC ("Send new RREQ for outbound packet to " <<header.GetDestination ());
          SendRequest (header.GetDestination ());
        }
    }
}

bool
RoutingProtocol::RouteInput (Ptr<const Packet> p, const Ipv4Header &header,
                             Ptr<const NetDevice> idev, UnicastForwardCallback ucb,
                             MulticastForwardCallback mcb, LocalDeliverCallback lcb, ErrorCallback ecb)
{
  NS_LOG_FUNCTION (taisba << p->GetUid () << header.GetDestination () << idev->GetAddress ());
  if (m_socketAddresses.empty ())
    {
      NS_LOG_LOGIC ("No aisba interfaces");
      return false;
    }
  NS_ASSERT (m_ipv4 != 0);
  NS_ASSERT (p != 0);
  // Check if input device supports IP
  NS_ASSERT (m_ipv4->GetInterfaceForDevice (idev) >= 0);
  int32_t iif = m_ipv4->GetInterfaceForDevice (idev);

  Ipv4Address dst = header.GetDestination ();
  Ipv4Address origin = header.GetSource ();

  // Deferred route request
  if (idev == m_lo)
    {
      DeferredRouteOutputTag tag;
      if (p->PeekPacketTag (tag))
        {
          DeferredRouteOutput (p, header, ucb, ecb);
          return true;
```

```
    }
  }

// Duplicate of own packet
if (IsMyOwnAddress (origin))
//std::cout<<origin<<"duplicate";
  return true;

// AISBA is not a multicast routing protocol
if (dst.IsMulticast ())
  {
   return false;
  }

// Broadcast local delivery/forwarding
for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
       m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
  {
   Ipv4InterfaceAddress iface = j->second;
   if (m_ipv4->GetInterfaceForAddress (iface.GetLocal ()) == iif)
    if (dst == iface.GetBroadcast () || dst.IsBroadcast ())
      {
       if (m_dpd.IsDuplicate (p, header))
         {
          NS_LOG_DEBUG ("Duplicated packet " << p->GetUid () << " from " << origin << ". Drop.");
          return true;
         }
       UpdateRouteLifeTime (origin, ActiveRouteTimeout);
       Ptr<Packet> packet = p->Copy ();
       if (lcb.IsNull () == false)
         {
          NS_LOG_LOGIC ("Broadcast local delivery to " << iface.GetLocal ());
          lcb (p, header, iif);
          // Fall through to additional processing
         }
       else
         {
          NS_LOG_ERROR ("Unable to deliver packet locally due to null callback " << p->GetUid () << "
from " << origin);
          ecb (p, header, Socket::ERROR_NOROUTETOHOST);
         }
       if (!EnableBroadcast)
         {
          return true;
         }
       if (header.GetTtl () > 1)
         {
          NS_LOG_LOGIC ("Forward broadcast. TTL " << (uint16_t) header.GetTtl ());
          RoutingTableEntry toBroadcast;
          if (m_routingTable.LookupRoute (dst, toBroadcast))
            {
             Ptr<Ipv4Route> route = toBroadcast.GetRoute ();
             ucb (route, packet, header);
```

```
              }
            else
              {
                NS_LOG_DEBUG ("No route to forward broadcast. Drop packet " << p->GetUid ());
              }
          }
        else
          {
            NS_LOG_DEBUG ("TTL exceeded. Drop packet " << p->GetUid ());
          }
        return true;
      }
  }

 // Unicast local delivery
 if (m_ipv4->IsDestinationAddress (dst, iif))
   {
    UpdateRouteLifeTime (origin, ActiveRouteTimeout);
    RoutingTableEntry toOrigin;
    if (m_routingTable.LookupValidRoute (origin, toOrigin))
      {
        UpdateRouteLifeTime (toOrigin.GetNextHop (), ActiveRouteTimeout);
        m_nb.Update (toOrigin.GetNextHop (), ActiveRouteTimeout);
      }
    if (lcb.IsNull () == false)
      {
        NS_LOG_LOGIC ("Unicast local delivery to " << dst);
        lcb (p, header, iif);
      }
    else
      {
        NS_LOG_ERROR ("Unable to deliver packet locally due to null callback " << p->GetUid () << " from "
<< origin);
        ecb (p, header, Socket::ERROR_NOROUTETOHOST);
      }
    return true;
   }

 // Forwarding
 return Forwarding (p, header, ucb, ecb);
}

bool
RoutingProtocol::Forwarding (Ptr<const Packet> p, const Ipv4Header & header,
                             UnicastForwardCallback ucb, ErrorCallback ecb)
{
 NS_LOG_FUNCTION (taisba);
 Ipv4Address dst = header.GetDestination ();
 Ipv4Address origin = header.GetSource ();
 m_routingTable.Purge ();
 RoutingTableEntry toDst;
 if (m_routingTable.LookupRoute (dst, toDst))
   {
```

```
    if (toDst.GetFlag () == VALID)

      {
       Ptr<Ipv4Route> route = toDst.GetRoute ();
       NS_LOG_LOGIC (route->GetSource ()<<" forwarding to " << dst << " from " << origin << " packet " <<
p->GetUid ());
   //  std::cout<< " all good";//lincy
       /*
        *  Each time a route is used to forward a data packet, its Active Route
        *  Lifetime field of the source, destination and the next hop on the
        *  path to the destination is updated to be no less than the current
        *  time plus ActiveRouteTimeout.
        */
       UpdateRouteLifeTime (origin, ActiveRouteTimeout);
       UpdateRouteLifeTime (dst, ActiveRouteTimeout);
       UpdateRouteLifeTime (route->GetGateway (), ActiveRouteTimeout);
       /*
        *  Since the route between each originator and destination pair is expected to be symmetric, the
        *  Active Route Lifetime for the previous hop, along the reverse path back to the IP source, is also
updated
        *  to be no less than the current time plus ActiveRouteTimeout
        */
       RoutingTableEntry toOrigin;
       m_routingTable.LookupRoute (origin, toOrigin);
       UpdateRouteLifeTime (toOrigin.GetNextHop (), ActiveRouteTimeout);

       m_nb.Update (route->GetGateway (), ActiveRouteTimeout);
       m_nb.Update (toOrigin.GetNextHop (), ActiveRouteTimeout);

       ucb (route, p, header);
       //sendPAMPSignal ( dst, route);

       return true;
      }
    else
      {
       if (toDst.GetValidSeqNo ())
         {
          SendRerrWhenNoRouteToForward (dst, toDst.GetSeqNo (), origin);
          std::cout<<"RERRWHENNOROUTETO FORWARD"<<dst<<origin<<"\n";//lincy
          //sendPAMPSignal(dst, origin);//lincy


          NS_LOG_DEBUG ("Drop packet " << p->GetUid () << " because no route to forward it.");
          return false;
         }
      }
  }
 NS_LOG_LOGIC ("route not found to "<< dst << ". Send RERR message.");

 NS_LOG_DEBUG ("Drop packet " << p->GetUid () << " because no route to forward it.");
 SendRerrWhenNoRouteToForward (dst, 0, origin);
 return false;
```

```
}

void
RoutingProtocol::SetIpv4 (Ptr<Ipv4> ipv4)
{
  NS_ASSERT (ipv4 != 0);
  NS_ASSERT (m_ipv4 == 0);

  m_ipv4 = ipv4;

  // Create lo route. It is asserted that the only one interface up for now is loopback
  NS_ASSERT (m_ipv4->GetNInterfaces () == 1 && m_ipv4->GetAddress (0, 0).GetLocal () == Ipv4Address
("127.0.0.1"));
  m_lo = m_ipv4->GetNetDevice (0);
  NS_ASSERT (m_lo != 0);
  // Remember lo route
  RoutingTableEntry rt (/*device=*/ m_lo, /*dst=*/ Ipv4Address::GetLoopback (), /*know seqno=*/ true,
/*seqno=*/ 0,
                        /*iface=*/ Ipv4InterfaceAddress (Ipv4Address::GetLoopback (), Ipv4Mask
("255.0.0.0")),
                        /*hops=*/ 1, /*next hop=*/ Ipv4Address::GetLoopback (),
                        /*lifetime=*/ Simulator::GetMaximumSimulationTime ());
  m_routingTable.AddRoute (rt);

  Simulator::ScheduleNow (&RoutingProtocol::Start, taisba);
}

void
RoutingProtocol::NotifyInterfaceUp (uint32_t i)
{
  // Lincy    Begins
  Ptr<Ipv4L3Protocol> l3K = m_ipv4->GetObject<Ipv4L3Protocol> ();
  Ipv4InterfaceAddress ifaceLocal = l3K->GetAddress(i,0);
  std::cout << ifaceLocal.GetLocal () << ": NODES Printing interface Is up\n";
  // Lincy Ends

  NS_LOG_FUNCTION (taisba << m_ipv4->GetAddress (i, 0).GetLocal ());
  Ptr<Ipv4L3Protocol> l3 = m_ipv4->GetObject<Ipv4L3Protocol> ();
  if (l3->GetNAddresses (i) > 1)
    {
      NS_LOG_WARN ("AISBA does not work with more then one address per each interface.");
    }
  Ipv4InterfaceAddress iface = l3->GetAddress (i, 0);
  if (iface.GetLocal () == Ipv4Address ("127.0.0.1"))
    return;

  // Create a socket to listen only on taisba interface
  Ptr<Socket> socket = Socket::CreateSocket (GetObject<Node> (),
                                             UdpSocketFactory::GetTypeId ());
  NS_ASSERT (socket != 0);
  socket->SetRecvCallback (MakeCallback (&RoutingProtocol::RecvAisba, taisba));
  socket->Bind (InetSocketAddress (Ipv4Address::GetAny (), AISBA_PORT));
  socket->BindToNetDevice (l3->GetNetDevice (i));
```

```
  socket->SetAllowBroadcast (true);
  socket->SetAttribute ("IpTtl", UintegerValue (1));
  m_socketAddresses.insert (std::make_pair (socket, iface));

  // create also a subnet broadcast socket
  socket = Socket::CreateSocket (GetObject<Node> (),
                                 UdpSocketFactory::GetTypeId ());
  NS_ASSERT (socket != 0);
  socket->SetRecvCallback (MakeCallback (&RoutingProtocol::RecvAisba, taisba));
  socket->Bind (InetSocketAddress (iface.GetBroadcast (), AISBA_PORT));
  socket->BindToNetDevice (l3->GetNetDevice (i));
  socket->SetAllowBroadcast (true);
  socket->SetAttribute ("IpTtl", UintegerValue (1));
  m_socketSubnetBroadcastAddresses.insert (std::make_pair (socket, iface));

  // Add local broadcast record to the routing table
  Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (iface.GetLocal ()));
  RoutingTableEntry rt (/*device=*/ dev, /*dst=*/ iface.GetBroadcast (), /*know seqno=*/ true, /*seqno=*/ 0,
/*iface=*/ iface,
                                  /*hops=*/ 1, /*next hop=*/ iface.GetBroadcast (), /*lifetime=*/
Simulator::GetMaximumSimulationTime ());
  m_routingTable.AddRoute (rt);

  if (l3->GetInterface (i)->GetArpCache ())
    {
      m_nb.AddArpCache (l3->GetInterface (i)->GetArpCache ());
    }

  // Allow neighbor manager use taisba interface for layer 2 feedback if possible
  Ptr<WifiNetDevice> wifi = dev->GetObject<WifiNetDevice> ();
  if (wifi == 0)
    return;
  Ptr<WifiMac> mac = wifi->GetMac ();
  if (mac == 0)
    return;

  mac->TraceConnectWithoutContext ("TxErrHeader", m_nb.GetTxErrorCallback ());
}

void
RoutingProtocol::NotifyInterfaceDown (uint32_t i)
{
  //Lincy
  std::cout << "Lincy Printing LInk Is Donw";

  NS_LOG_FUNCTION (taisba << m_ipv4->GetAddress (i, 0).GetLocal ());

  // Disable layer 2 link state monitoring (if possible)
  Ptr<Ipv4L3Protocol> l3 = m_ipv4->GetObject<Ipv4L3Protocol> ();
  Ptr<NetDevice> dev = l3->GetNetDevice (i);
  Ptr<WifiNetDevice> wifi = dev->GetObject<WifiNetDevice> ();
  if (wifi != 0)
    {
```

```
    Ptr<WifiMac> mac = wifi->GetMac ()->GetObject<AdhocWifiMac> ();
    if (mac != 0)
      {
       mac->TraceDisconnectWithoutContext ("TxErrHeader",
                                           m_nb.GetTxErrorCallback ());
       m_nb.DelArpCache (l3->GetInterface (i)->GetArpCache ());
      }
   }

  // Close socket
  Ptr<Socket> socket = FindSocketWithInterfaceAddress (m_ipv4->GetAddress (i, 0));
  NS_ASSERT (socket);
  socket->Close ();
  m_socketAddresses.erase (socket);

  // Close socket
  socket = FindSubnetBroadcastSocketWithInterfaceAddress (m_ipv4->GetAddress (i, 0));
  NS_ASSERT (socket);
  socket->Close ();
  m_socketSubnetBroadcastAddresses.erase (socket);

  if (m_socketAddresses.empty ())
    {
     NS_LOG_LOGIC ("No aisba interfaces");
     m_htimer.Cancel ();
     m_nb.Clear ();
     m_routingTable.Clear ();
     return;
    }
  m_routingTable.DeleteAllRoutesFromInterface (m_ipv4->GetAddress (i, 0));
}

void
RoutingProtocol::NotifyAddAddress (uint32_t i, Ipv4InterfaceAddress address)
{
  NS_LOG_FUNCTION (taisba << " interface " << i << " address " << address);
  Ptr<Ipv4L3Protocol> l3 = m_ipv4->GetObject<Ipv4L3Protocol> ();
  if (!l3->IsUp (i))
    return;
  if (l3->GetNAddresses (i) == 1)
    {
     Ipv4InterfaceAddress iface = l3->GetAddress (i, 0);
     Ptr<Socket> socket = FindSocketWithInterfaceAddress (iface);
     if (!socket)
       {
        if (iface.GetLocal () == Ipv4Address ("127.0.0.1"))
          return;
        // Create a socket to listen only on taisba interface
        Ptr<Socket> socket = Socket::CreateSocket (GetObject<Node> (),
                                           UdpSocketFactory::GetTypeId ());
        NS_ASSERT (socket != 0);
        socket->SetRecvCallback (MakeCallback (&RoutingProtocol::RecvAisba,taisba));
        socket->Bind (InetSocketAddress (iface.GetLocal (), AISBA_PORT));
```

```
        socket->BindToNetDevice (l3->GetNetDevice (i));
        socket->SetAllowBroadcast (true);
        m_socketAddresses.insert (std::make_pair (socket, iface));

        // create also a subnet directed broadcast socket
        socket = Socket::CreateSocket (GetObject<Node> (),
                                       UdpSocketFactory::GetTypeId ());
        NS_ASSERT (socket != 0);
        socket->SetRecvCallback (MakeCallback (&RoutingProtocol::RecvAisba, taisba));
        socket->Bind (InetSocketAddress (iface.GetBroadcast (), AISBA_PORT));
        socket->BindToNetDevice (l3->GetNetDevice (i));
        socket->SetAllowBroadcast (true);
        socket->SetAttribute ("IpTtl", UintegerValue (1));
        m_socketSubnetBroadcastAddresses.insert (std::make_pair (socket, iface));

        // Add local broadcast record to the routing table
        Ptr<NetDevice> dev = m_ipv4->GetNetDevice (
          m_ipv4->GetInterfaceForAddress (iface.GetLocal ()));
        RoutingTableEntry rt (/*device=*/ dev, /*dst=*/ iface.GetBroadcast (), /*know seqno=*/ true,
                              /*seqno=*/ 0, /*iface=*/ iface, /*hops=*/ 1,
                              /*next hop=*/ iface.GetBroadcast (), /*lifetime=*/
Simulator::GetMaximumSimulationTime ());
        m_routingTable.AddRoute (rt);
      }
    }
  else
    {
      NS_LOG_LOGIC ("AISBA does not work with more then one address per each interface. Ignore added
address");
    }
}

void
RoutingProtocol::NotifyRemoveAddress (uint32_t i, Ipv4InterfaceAddress address)
{
  NS_LOG_FUNCTION (taisba);
  Ptr<Socket> socket = FindSocketWithInterfaceAddress (address);
  if (socket)
    {
      m_routingTable.DeleteAllRoutesFromInterface (address);
      socket->Close ();
      m_socketAddresses.erase (socket);

      Ptr<Socket> unicastSocket = FindSubnetBroadcastSocketWithInterfaceAddress (address);
      if (unicastSocket)
        {
          unicastSocket->Close ();
          m_socketAddresses.erase (unicastSocket);
        }

      Ptr<Ipv4L3Protocol> l3 = m_ipv4->GetObject<Ipv4L3Protocol> ();
      if (l3->GetNAddresses (i))
        {
```

```
        Ipv4InterfaceAddress iface = l3->GetAddress (i, 0);
        // Create a socket to listen only on taisba interface
        Ptr<Socket> socket = Socket::CreateSocket (GetObject<Node> (),
                                           UdpSocketFactory::GetTypeId ());
        NS_ASSERT (socket != 0);
        socket->SetRecvCallback (MakeCallback (&RoutingProtocol::RecvAisba, taisba));
        // Bind to any IP address so that broadcasts can be received
        socket->Bind (InetSocketAddress (iface.GetLocal (), AISBA_PORT));
        socket->BindToNetDevice (l3->GetNetDevice (i));
        socket->SetAllowBroadcast (true);
        socket->SetAttribute ("IpTtl", UintegerValue (1));
        m_socketAddresses.insert (std::make_pair (socket, iface));

        // create also a unicast socket
        socket = Socket::CreateSocket (GetObject<Node> (),
                                           UdpSocketFactory::GetTypeId ());
        NS_ASSERT (socket != 0);
        socket->SetRecvCallback (MakeCallback (&RoutingProtocol::RecvAisba, taisba));
        socket->Bind (InetSocketAddress (iface.GetBroadcast (), AISBA_PORT));
        socket->BindToNetDevice (l3->GetNetDevice (i));
        socket->SetAllowBroadcast (true);
        socket->SetAttribute ("IpTtl", UintegerValue (1));
        m_socketSubnetBroadcastAddresses.insert (std::make_pair (socket, iface));

        // Add local broadcast record to the routing table
        Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (iface.GetLocal ()));
        RoutingTableEntry rt (/*device=*/ dev, /*dst=*/ iface.GetBroadcast (), /*know seqno=*/ true, /*seqno=*/
0, /*iface=*/ iface,
                                  /*hops=*/ 1, /*next hop=*/ iface.GetBroadcast (), /*lifetime=*/
Simulator::GetMaximumSimulationTime ());
        m_routingTable.AddRoute (rt);
       }
     if (m_socketAddresses.empty ())
       {
        NS_LOG_LOGIC ("No aisba interfaces");
        m_htimer.Cancel ();
        m_nb.Clear ();
        m_routingTable.Clear ();
        return;
       }
   }
  else
   {
     NS_LOG_LOGIC ("Remove address not participating in AISBA operation");
   }
}

bool
RoutingProtocol::IsMyOwnAddress (Ipv4Address src)
{
  NS_LOG_FUNCTION (taisba << src);
  for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
        m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
```

```
    {
     Ipv4InterfaceAddress iface = j->second;
     //std::cout << "From isMyOwnAddress Method: "<<iface.GetLocal();
     if (src == iface.GetLocal ())
       {
        return true;
       }
    }
  return false;
}

Ptr<Ipv4Route>
RoutingProtocol::LoopbackRoute (const Ipv4Header & hdr, Ptr<NetDevice> oif) const
{
  NS_LOG_FUNCTION (taisba << hdr);
  NS_ASSERT (m_lo != 0);
  Ptr<Ipv4Route> rt = Create<Ipv4Route> ();
  rt->SetDestination (hdr.GetDestination ());
  //
  // Source address selection here is tricky.  The loopback route is
  // returned when AISBA does not have a route; taisba causes the packet
  // to be looped back and handled (cached) in RouteInput() method
  // while a route is found. However, connection-oriented protocols
  // like TCP need to create an endpoint four-tuple (src, src port,
  // dst, dst port) and create a pseudo-header for checksumming.  So,
  // AISBA needs to guess correctly what the eventual source address
  // will be.
  //
  // For single interface, single address nodes, taisba is not a problem.
  // When there are possibly multiple outgoing interfaces, the policy
  // implemented here is to pick the first available AISBA interface.
  // If RouteOutput() caller specified an outgoing interface, that
  // further constrains the selection of source address
  //
  std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j = m_socketAddresses.begin ();
  if (oif)
    {
     // Iterate to find an address on the oif device
     for (j = m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
       {
        Ipv4Address addr = j->second.GetLocal ();
        int32_t interface = m_ipv4->GetInterfaceForAddress (addr);
        if (oif == m_ipv4->GetNetDevice (static_cast<uint32_t> (interface)))
          {
           rt->SetSource (addr);
           break;
          }
       }
    }
  else
    {
     rt->SetSource (j->second.GetLocal ());
    }
```

```
  NS_ASSERT_MSG (rt->GetSource () != Ipv4Address (), "Valid AISBA source address not found");
  rt->SetGateway (Ipv4Address ("127.0.0.1"));
  rt->SetOutputDevice (m_lo);
  return rt;
}

void
RoutingProtocol::SendRequest (Ipv4Address dst)
{
  NS_LOG_FUNCTION ( taisba << dst);
  // A node SHOULD NOT originate more than RREQ_RATELIMIT RREQ messages per second.
  if (m_rreqCount == RreqRateLimit)
    {
      Simulator::Schedule (m_rreqRateLimitTimer.GetDelayLeft () + MicroSeconds (100),
                           &RoutingProtocol::SendRequest, taisba, dst);

      return;
    }
  else
    m_rreqCount++;
   // std::cout<< m_rreqCount;//lincy
  // Create RREQ header
  RreqHeader rreqHeader;
  rreqHeader.SetDst (dst);

  RoutingTableEntry rt;
  if (m_routingTable.LookupRoute (dst, rt))
    {
      rreqHeader.SetHopCount (rt.GetHop ());
      if (rt.GetValidSeqNo ())
        rreqHeader.SetDstSeqno (rt.GetSeqNo ());
      else
        rreqHeader.SetUnknownSeqno (true);
      rt.SetFlag (IN_SEARCH);
      m_routingTable.Update (rt);
      // Lincy Starts
      Ptr<Ipv4L3Protocol> l3K = m_ipv4->GetObject<Ipv4L3Protocol> ();
      Ipv4InterfaceAddress ifaceLocal = l3K->GetAddress(1,0);
      std::cout << ifaceLocal.GetLocal() <<": insearch-update routing table entry. \n";//lincy
      //Lincy Ends
    }
  else
    {
      rreqHeader.SetUnknownSeqno (true);
      Ptr<NetDevice> dev = 0;
      RoutingTableEntry newEntry (/*device=*/ dev, /*dst=*/ dst, /*validSeqNo=*/ false, /*seqno=*/ 0,
                                  /*iface=*/ Ipv4InterfaceAddress (),/*hop=*/ 0,
                                  /*nextHop=*/ Ipv4Address (), /*lifeTime=*/ Seconds (0));
      newEntry.SetFlag (IN_SEARCH);
      m_routingTable.AddRoute (newEntry);
      //Lincy Starts
      Ptr<Ipv4L3Protocol> l3K = m_ipv4->GetObject<Ipv4L3Protocol> ();
      Ipv4InterfaceAddress ifaceLocal = l3K->GetAddress(1,0);
```

```
    std::cout<< ifaceLocal.GetLocal() <<": add new entry to routing table.\n";//lincy
    //Lincy Ends
   }

 if (GratuitousReply)
   rreqHeader.SetGratiousRrep (true);
 if (DestinationOnly)
   rreqHeader.SetDestinationOnly (true);

 m_seqNo++;
 rreqHeader.SetOriginSeqno (m_seqNo);
 m_requestId++;
 rreqHeader.SetId (m_requestId);
 rreqHeader.SetHopCount (0);

 // Send RREQ as subnet directed broadcast from each interface used by aisba
 for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
       m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
   {
     Ptr<Socket> socket = j->first;
     Ipv4InterfaceAddress iface = j->second;

     rreqHeader.SetOrigin (iface.GetLocal ());
     m_rreqIdCache.IsDuplicate (iface.GetLocal (), m_requestId);

     Ptr<Packet> packet = Create<Packet> ();
     packet->AddHeader (rreqHeader);
     TypeHeader tHeader (AISBATYPE_RREQ);
     packet->AddHeader (tHeader);
     // Send to all-hosts broadcast if on /32 addr, subnet-directed otherwise
     Ipv4Address destination;
     if (iface.GetMask () == Ipv4Mask::GetOnes ())
       {
         destination = Ipv4Address ("255.255.255.255");
       }
     else
       {
         destination = iface.GetBroadcast ();
       }
     NS_LOG_DEBUG ("Send RREQ with id " << rreqHeader.GetId () << " to socket");
     m_lastBcastTime = Simulator::Now ();
     Simulator::Schedule (Time (MilliSeconds (m_uniformRandomVariable->GetInteger (0, 10))),
&RoutingProtocol::SendTo, taisba, socket, packet, destination);
    }
 //  ackPAMPSignal (rreqHeader,toOrigin);
  ScheduleRreqRetry (dst);
  std::cout<<dst<<"destination";//lincy
}//send rreq ends here//lincy

void
RoutingProtocol::SendTo (Ptr<Socket> socket, Ptr<Packet> packet, Ipv4Address destination)
{
   socket->SendTo (packet, 0, InetSocketAddress (destination, AISBA_PORT));
```

```
}
void
RoutingProtocol::ScheduleRreqRetry (Ipv4Address dst)
{
  NS_LOG_FUNCTION (taisba << dst);
  std::cout<<dst<<"i found you destination";
  if (m_addressReqTimer.find (dst) == m_addressReqTimer.end ())
    {
      Timer timer (Timer::CANCEL_ON_DESTROY);
      m_addressReqTimer[dst] = timer;
    }
  m_addressReqTimer[dst].SetFunction (&RoutingProtocol::RouteRequestTimerExpire, taisba);
  m_addressReqTimer[dst].Remove ();
  m_addressReqTimer[dst].SetArguments (dst);
  RoutingTableEntry rt;
  m_routingTable.LookupRoute (dst, rt);
  rt.IncrementRreqCnt ();
  m_routingTable.Update (rt);
  m_addressReqTimer[dst].Schedule (Time (rt.GetRreqCnt () * NetTraversalTime));
  NS_LOG_LOGIC ("Scheduled RREQ retry in " << Time (rt.GetRreqCnt () * NetTraversalTime).GetSeconds
() << " seconds");
}

void
RoutingProtocol::RecvAisba (Ptr<Socket> socket)
{  //RerrHeader rerrHeader;//LINCY

  NS_LOG_FUNCTION (taisba << socket);
  Address sourceAddress;
  Ptr<Packet> packet = socket->RecvFrom (sourceAddress);
  InetSocketAddress inetSourceAddr = InetSocketAddress::ConvertFrom (sourceAddress);
  Ipv4Address sender = inetSourceAddr.GetIpv4 ();
  Ipv4Address receiver;

  if (m_socketAddresses.find (socket) != m_socketAddresses.end ())
    {
      receiver = m_socketAddresses[socket].GetLocal ();
    }
  else if(m_socketSubnetBroadcastAddresses.find (socket) != m_socketSubnetBroadcastAddresses.end ())
    {
      receiver = m_socketSubnetBroadcastAddresses[socket].GetLocal ();
    }
  else
    {
      NS_ASSERT_MSG (false, "Received a packet from an unknown socket");
    }
  NS_LOG_DEBUG ("AISBA node " << taisba << " received a AISBA packet from " << sender << " to " <<
receiver);

  UpdateRouteToNeighbor (sender, receiver);
  TypeHeader tHeader (AISBATYPE_RREQ);
  packet->RemoveHeader (tHeader);
```

```
 if (!tHeader.IsValid ())
   {
     NS_LOG_DEBUG ("AISBA message " << packet->GetUid () << " with unknown type received: " <<
tHeader.Get () << ". Drop");
     return; // drop
   }
 switch (tHeader.Get ())
   {
   case AISBATYPE_RREQ:
     {
      RecvRequest (packet, receiver, sender);
       std::cout<<"RREQPACKET :"<<packet<<" receiver :"<<receiver <<" sender :"<<sender<<"\n";
      //std::cout<<"RREQ.\n"; //LINCY
    //  std::cout<< m_rreqCount;
      break;
     }
   case AISBATYPE_RREP:
     {
      RecvReply (packet, receiver, sender);
      std::cout<<"RREPPACKET :"<<packet<<" receiver :"<<receiver <<" sender :"<<sender<<"\n";



      break;
     }
   case AISBATYPE_RERR:
     {
      RecvError (packet, sender);
     std::cout<<sender<<":SENDER OF RERR \n";//lincy
     // RerrHeader rerrHeader;

 ////p->RemoveHeader (rerrHeader);
//// PampsendHeader pampsendHeader;

              ////Lincy Starts

               //Ptr<Ipv4L3Protocol> l3K = m_ipv4->GetObject<Ipv4L3Protocol> ();
               //Ipv4InterfaceAddress ifaceLocal = l3K->GetAddress(1, 0);

             ///**lINCY  ADDED code to send PAMP when source has received RERR,ALSO ADDED
GETORIGIN,SET ORIGIN TO PACKET.CC AND H FOR TAISBA
PURPOSE*************************************************************/

     //std::cout <<  " RERR Packet Received TO node - " << ifaceLocal.GetLocal() << "\n";//lincy

          //rerrHeader.SetOrigin (ifaceLocal.GetLocal ());
           //Ipv4Address origin=rerrHeader.GetOrigin ();


        //if (IsMyOwnAddress (rerrHeader.GetOrigin())){
                                              //std::cout<<" Calling Activate DC for " <<origin;
                                               //ActivateDC();//Lincy
```

```
                        //}
      //Ipv4Address dst = rerrHeader.GetDst();
      //RoutingTableEntry toDst;
      //Ptr<Ipv4Route> route = toDst.GetRoute ();
        //sendPAMPSignal ( dst, route);
       ////ENDS LINCY

       break;
      }
    case AISBATYPE_RREP_ACK:
      {
       RecvReplyAck (sender);
       std::cout<<"RECVREPLYACK";//LINCY

       break;
      }
        case  AISBATYPE_PAMPSEND: ///ADD LINCY
    {
                        std::cout << "PAMP_SEND";
                        break;
                }
                 case  AISBATYPE_PAMPRECV: ///ADD LINCY
    {
                        std::cout << "PAMP_RECV";
                        break;
                }
    }
}

bool
RoutingProtocol::UpdateRouteLifeTime (Ipv4Address addr, Time lifetime)
{
  NS_LOG_FUNCTION (taisba << addr << lifetime);
  RoutingTableEntry rt;
  if (m_routingTable.LookupRoute (addr, rt))
    {
      if (rt.GetFlag () == VALID)
        {
          NS_LOG_DEBUG ("Updating VALID route");
          rt.SetRreqCnt (0);
          rt.SetLifeTime (std::max (lifetime, rt.GetLifeTime ()));
          m_routingTable.Update (rt);
          return true;
        }
    }
  return false;
}

void
RoutingProtocol::UpdateRouteToNeighbor (Ipv4Address sender, Ipv4Address receiver)
{
  NS_LOG_FUNCTION (taisba << "sender " << sender << " receiver " << receiver);
  RoutingTableEntry toNeighbor;
```

```
  if (!m_routingTable.LookupRoute (sender, toNeighbor))
    {
      Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
      RoutingTableEntry newEntry (/*device=*/ dev, /*dst=*/ sender, /*know seqno=*/ false, /*seqno=*/ 0,
                                  /*iface=*/ m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0),
                                  /*hops=*/ 1, /*next hop=*/ sender, /*lifetime=*/ ActiveRouteTimeout);
      m_routingTable.AddRoute (newEntry);
    }
  else
    {
      Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
      if (toNeighbor.GetValidSeqNo () && (toNeighbor.GetHop () == 1) && (toNeighbor.GetOutputDevice () ==
dev))
        {
          toNeighbor.SetLifeTime (std::max (ActiveRouteTimeout, toNeighbor.GetLifeTime ()));
        }
      else
        {
          RoutingTableEntry newEntry (/*device=*/ dev, /*dst=*/ sender, /*know seqno=*/ false, /*seqno=*/ 0,
                                      /*iface=*/ m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver),
0),
                                      /*hops=*/ 1, /*next hop=*/ sender, /*lifetime=*/ std::max (ActiveRouteTimeout,
toNeighbor.GetLifeTime ()));
          m_routingTable.Update (newEntry);
        }
    }

}

void
RoutingProtocol::RecvRequest (Ptr<Packet> p, Ipv4Address receiver, Ipv4Address src)
{
  NS_LOG_FUNCTION (taisba);
  RreqHeader rreqHeader;
  p->RemoveHeader (rreqHeader);

  // A node ignores all RREQs received from any node in its blacklist
  RoutingTableEntry toPrev;
  if (m_routingTable.LookupRoute (src, toPrev))
    {
      if (toPrev.IsUnidirectional ())
        {
          NS_LOG_DEBUG ("Ignoring RREQ from node in blacklist");
          return;
        }
    }

  uint32_t id = rreqHeader.GetId ();
  Ipv4Address origin = rreqHeader.GetOrigin ();
std::cout<<origin<<" :ORIGIN\n";//lincy
  /*
   *  Node checks to determine whether it has received a RREQ with the same Originator IP Address and RREQ
ID.
```

```
   *  If such a RREQ has been received, the node silently discards the newly received RREQ.
   */
  if (m_rreqIdCache.IsDuplicate (origin, id))
    {
      NS_LOG_DEBUG ("Ignoring RREQ due to duplicate");
      return;
    }

  // Increment RREQ hop count
  uint8_t hop = rreqHeader.GetHopCount () + 1;
  rreqHeader.SetHopCount (hop);

  /*
   *  When the reverse route is created or updated, the following actions on the route are also carried out:
   *  1. the Originator Sequence Number from the RREQ is compared to the corresponding destination sequence
number
   *     in the route table entry and copied if greater than the existing value there
   *  2. the valid sequence number field is set to true;
   *  3. the next hop in the routing table becomes the node from which the  RREQ was received
   *  4. the hop count is copied from the Hop Count in the RREQ message;
   *  5. the Lifetime is set to be the maximum of (ExistingLifetime, MinimalLifetime), where
   *     MinimalLifetime = current time + 2*NetTraversalTime - 2*HopCount*NodeTraversalTime
   */
  RoutingTableEntry toOrigin;
  if (!m_routingTable.LookupRoute (origin, toOrigin))
    {
      Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
      RoutingTableEntry newEntry (/*device=*/ dev, /*dst=*/ origin, /*validSeno=*/ true, /*seqNo=*/
rreqHeader.GetOriginSeqno (),
                                  /*iface=*/ m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0),
/*hops=*/ hop,
                                  /*nextHop*/ src, /*timeLife=*/ Time ((2 * NetTraversalTime - 2 * hop *
NodeTraversalTime)));
      m_routingTable.AddRoute (newEntry);
    }
  else
    {
      if (toOrigin.GetValidSeqNo ())
        {
          if (int32_t (rreqHeader.GetOriginSeqno ()) - int32_t (toOrigin.GetSeqNo ()) > 0)
            toOrigin.SetSeqNo (rreqHeader.GetOriginSeqno ());
        }
      else
        toOrigin.SetSeqNo (rreqHeader.GetOriginSeqno ());
      toOrigin.SetValidSeqNo (true);
      toOrigin.SetNextHop (src);
      toOrigin.SetOutputDevice (m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver)));
      toOrigin.SetInterface (m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0));
      toOrigin.SetHop (hop);
      toOrigin.SetLifeTime (std::max (Time (2 * NetTraversalTime - 2 * hop * NodeTraversalTime),
                                      toOrigin.GetLifeTime ()));
      m_routingTable.Update (toOrigin);
      //m_nb.Update (src, Time (AllowedHelloLoss * HelloInterval));
```

```
  }


 RoutingTableEntry toNeighbor;
 if (!m_routingTable.LookupRoute (src, toNeighbor))
  {
   NS_LOG_DEBUG ("Neighbor:" << src << " not found in routing table. Creating an entry");
   Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
   RoutingTableEntry newEntry (dev, src, false, rreqHeader.GetOriginSeqno (),
                               m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0),
                               1, src, ActiveRouteTimeout);
   m_routingTable.AddRoute (newEntry);
  }
 else
  {
   toNeighbor.SetLifeTime (ActiveRouteTimeout);
   toNeighbor.SetValidSeqNo (false);
   toNeighbor.SetSeqNo (rreqHeader.GetOriginSeqno ());
   toNeighbor.SetFlag (VALID);// IF INVALID THEN 0 PACKET RECEIVED IN PING STATISTICS
LINCY
   toNeighbor.SetOutputDevice (m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver)));
   toNeighbor.SetInterface (m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0));
   toNeighbor.SetHop (1);
   toNeighbor.SetNextHop (src);
   m_routingTable.Update (toNeighbor);
  }
 m_nb.Update (src, Time (AllowedHelloLoss * HelloInterval));

 NS_LOG_LOGIC (receiver << " receive RREQ with hop count " <<
static_cast<uint32_t>(rreqHeader.GetHopCount ())
                  << " ID " << rreqHeader.GetId ()
                  << " to destination " << rreqHeader.GetDst ());

 //  A node generates a RREP if either:
 //  (i)  it is itself the destination,
 if (IsMyOwnAddress (rreqHeader.GetDst ()))
  {
   m_routingTable.LookupRoute (origin, toOrigin);

   NS_LOG_DEBUG ("Send reply since I am the destination");//ORIGINAL
    SendReply (rreqHeader, toOrigin);//ORIGINAL
    //std::cout<<rreqHeader.GetOrigin()<<"HEADER";

    //TO CONFIRM DESTINATION GOT PAMP SIGNAL //LINCY
    // ackPAMPSignal (rreqHeader,toOrigin);


   return;
  }


 /*
  *
```

```
   * (ii) or it has an active route to the destination, the destination sequence number in the node's existing route
table entry for the destination
   *      is valid and greater than or equal to the Destination Sequence Number of the RREQ, and the "destination
only" flag is NOT set.
   */
  RoutingTableEntry toDst;
  Ipv4Address dst = rreqHeader.GetDst ();
  if (m_routingTable.LookupRoute (dst, toDst))
    {
     /*
      * Drop RREQ, Taisba node RREP wil make a loop.
      */
     if (toDst.GetNextHop () == src)
       {
        NS_LOG_DEBUG ("Drop RREQ from " << src << ", dest next hop " << toDst.GetNextHop ());
        return;
       }
     /*
      * The Destination Sequence number for the requested destination is set to the maximum of the
corresponding value
      * received in the RREQ message, and the destination sequence value currently maintained by the node for
the requested destination.
      * However, the forwarding node MUST NOT modify its maintained value for the destination sequence
number, even if the value
      * received in the incoming RREQ is larger than the value currently maintained by the forwarding node.
      */
     if ((rreqHeader.GetUnknownSeqno () || (int32_t (toDst.GetSeqNo ()) - int32_t (rreqHeader.GetDstSeqno ())
>= 0))
         && toDst.GetValidSeqNo () )
       {
        if (!rreqHeader.GetDestinationOnly () && toDst.GetFlag () == INVALID)
          {
           m_routingTable.LookupRoute (origin, toOrigin);
           SendReplyByIntermediateNode (toDst, toOrigin, rreqHeader.GetGratiousRrep ());
           return;
          }
        rreqHeader.SetDstSeqno (toDst.GetSeqNo ());
        rreqHeader.SetUnknownSeqno (false);
       }
    }

  for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
       m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
    {
     Ptr<Socket> socket = j->first;
     Ipv4InterfaceAddress iface = j->second;
     Ptr<Packet> packet = Create<Packet> ();
     packet->AddHeader (rreqHeader);
     TypeHeader tHeader (AISBATYPE_RREQ);
     packet->AddHeader (tHeader);
     // Send to all-hosts broadcast if on /32 addr, subnet-directed otherwise
     Ipv4Address destination;
     if (iface.GetMask () == Ipv4Mask::GetOnes ())
```

```
      {
        destination = Ipv4Address ("255.255.255.255");
      }
    else
      {
        destination = iface.GetBroadcast ();
      }
    m_lastBcastTime = Simulator::Now ();
    Simulator::Schedule (Time (MilliSeconds (m_uniformRandomVariable->GetInteger (0, 10))),
&RoutingProtocol::SendTo, taisba, socket, packet, destination);

   }
}

void
RoutingProtocol::SendReply (RreqHeader const & rreqHeader, RoutingTableEntry const & toOrigin)
{
 NS_LOG_FUNCTION (taisba << toOrigin.GetDestination ());

 std::cout<<"i am source to which dest should send rrep" <<toOrigin.GetDestination() << "\n";//lincy

 /*
  * Destination node MUST increment its own sequence number by one if the sequence number in the RREQ
packet is equal to that
  * incremented value. Otherwise, the destination does not change its sequence number before generating the
RREP message.
  */
 if (!rreqHeader.GetUnknownSeqno () && (rreqHeader.GetDstSeqno () == m_seqNo + 1))
   m_seqNo++;
 RrepHeader rrepHeader ( /*prefixSize=*/ 0, /*hops=*/ 0, /*dst=*/ rreqHeader.GetDst (),
                         /*dstSeqNo=*/ m_seqNo, /*origin=*/ toOrigin.GetDestination (), /*lifeTime=*/
MyRouteTimeout);
 Ptr<Packet> packet = Create<Packet> ();
 packet->AddHeader (rrepHeader);
 TypeHeader tHeader (AISBATYPE_RREP);
 packet->AddHeader (tHeader);
 Ptr<Socket> socket = FindSocketWithInterfaceAddress (toOrigin.GetInterface ());
 NS_ASSERT (socket);
 socket->SendTo (packet, 0, InetSocketAddress (toOrigin.GetNextHop (), AISBA_PORT));
}

//method added by lincy for PAMPSEND should i follow the send request or send reply ,i am following send
reply because the path is already known inorder to send pamp

//void
//RoutingProtocol::PAMPSend (RreqHeader const & rreqHeader, RoutingTableEntry const & toOrigin)
//{
 //NS_LOG_FUNCTION (taisba << toOrigin.GetDestination ());

 //std::cout<<"i SHOULD SEND pamp to" <<rreqHeader.GetDst () << "\n";//lincy

 ///*
```

```
  //* Destination node MUST increment its own sequence number by one if the sequence number in the RREQ
packet is equal to that
  //* incremented value. Otherwise, the destination does not change its sequence number before generating the
RREP message.
  //*/
 //if (!rreqHeader.GetUnknownSeqno () && (rreqHeader.GetDstSeqno () == m_seqNo + 1))
   //m_seqNo++;
   //PampsendHeader pampsendheader ( /*prefixSize=*/ 0, /*hops=*/ 0, /*dst=*/ rreqHeader.GetDst (),
                                ///*dstSeqNo=*/ m_seqNo, /*origin=*/ toOrigin.GetDestination (), /*lifeTime=*/
MyRouteTimeout);
 //Ptr<Packet> packet = Create<Packet> ();
 ////PampsendHeader pampsendheader;
 //packet->AddHeader (pampsendheader);
 //TypeHeader tHeader (AISBATYPE_PAMPSEND);
 //packet->AddHeader (tHeader);
 //Ptr<Socket> socket = FindSocketWithInterfaceAddress (toOrigin.GetInterface ());
 //NS_ASSERT (socket);
 //socket->SendTo (packet, 0, InetSocketAddress (toOrigin.GetNextHop (), AISBA_PORT));
//}

void
RoutingProtocol::SendReplyByIntermediateNode (RoutingTableEntry & toDst, RoutingTableEntry & toOrigin,
bool gratRep)
{
 NS_LOG_FUNCTION (taisba);
 RrepHeader rrepHeader (/*prefix size=*/ 0, /*hops=*/ toDst.GetHop (), /*dst=*/ toDst.GetDestination (), /*dst
seqno=*/ toDst.GetSeqNo (),
                                /*origin=*/ toOrigin.GetDestination (), /*lifetime=*/ toDst.GetLifeTime ());
 /* If the node we received a RREQ for is a neighbor we are
  * probably facing a unidirectional link... Better request a RREP-ack
  */
 if (toDst.GetHop () == 1)
   {
     rrepHeader.SetAckRequired (true);
     RoutingTableEntry toNextHop;
     m_routingTable.LookupRoute (toOrigin.GetNextHop (), toNextHop);
     toNextHop.m_ackTimer.SetFunction (&RoutingProtocol::AckTimerExpire, taisba);
     toNextHop.m_ackTimer.SetArguments (toNextHop.GetDestination (), BlackListTimeout);
     toNextHop.m_ackTimer.SetDelay (NextHopWait);
   }
 toDst.InsertPrecursor (toOrigin.GetNextHop ());
 toOrigin.InsertPrecursor (toDst.GetNextHop ());
 m_routingTable.Update (toDst);
 m_routingTable.Update (toOrigin);


 Ptr<Packet> packet = Create<Packet> ();
 packet->AddHeader (rrepHeader);
 TypeHeader tHeader (AISBATYPE_RREP);
 packet->AddHeader (tHeader);
 Ptr<Socket> socket = FindSocketWithInterfaceAddress (toOrigin.GetInterface ());
 NS_ASSERT (socket);
 socket->SendTo (packet, 0, InetSocketAddress (toOrigin.GetNextHop (), AISBA_PORT));
```

```
  // Generating gratuitous RREPs
  if (gratRep)
    {
      RrepHeader gratRepHeader (/*prefix size=*/ 0, /*hops=*/ toOrigin.GetHop (), /*dst=*/
toOrigin.GetDestination (),
                                /*dst seqno=*/ toOrigin.GetSeqNo (), /*origin=*/ toDst.GetDestination (),
                                /*lifetime=*/ toOrigin.GetLifeTime ());
      Ptr<Packet> packetToDst = Create<Packet> ();
      packetToDst->AddHeader (gratRepHeader);
      TypeHeader type (AISBATYPE_RREP);
      packetToDst->AddHeader (type);
      Ptr<Socket> socket = FindSocketWithInterfaceAddress (toDst.GetInterface ());
      NS_ASSERT (socket);
      NS_LOG_LOGIC ("Send gratuitous RREP " << packet->GetUid ());
      socket->SendTo (packetToDst, 0, InetSocketAddress (toDst.GetNextHop (), AISBA_PORT));
    }
}

void
RoutingProtocol::SendReplyAck (Ipv4Address neighbor)
{
  NS_LOG_FUNCTION (taisba << " to " << neighbor);
  RrepAckHeader h;
  TypeHeader typeHeader (AISBATYPE_RREP_ACK);
  Ptr<Packet> packet = Create<Packet> ();
  packet->AddHeader (h);
  packet->AddHeader (typeHeader);
  RoutingTableEntry toNeighbor;
  m_routingTable.LookupRoute (neighbor, toNeighbor);
  Ptr<Socket> socket = FindSocketWithInterfaceAddress (toNeighbor.GetInterface ());
  NS_ASSERT (socket);
  socket->SendTo (packet, 0, InetSocketAddress (neighbor, AISBA_PORT));
}

void
RoutingProtocol::RecvReply (Ptr<Packet> p, Ipv4Address receiver, Ipv4Address sender)
{
  NS_LOG_FUNCTION (taisba << " src " << sender);
  //std::cout<<receiver<<"RECEIVER";
  RrepHeader rrepHeader;
  p->RemoveHeader (rrepHeader);
  Ipv4Address dst = rrepHeader.GetDst ();

  NS_LOG_LOGIC ("RREP destination " << dst << " RREP origin " << rrepHeader.GetOrigin ());


  uint8_t hop = rrepHeader.GetHopCount () + 1;
  rrepHeader.SetHopCount (hop);

  // If RREP is Hello message
  if (dst == rrepHeader.GetOrigin ())
    {
```

```
    ProcessHello (rrepHeader, receiver);
    return;
  }

 /*
  * If the route table entry to the destination is created or updated, then the following actions occur:
  * -  the route is marked as active,
  * -  the destination sequence number is marked as valid,
  * -  the next hop in the route entry is assigned to be the node from which the RREP is received,
  *    which is indicated by the source IP address field in the IP header,
  * -  the hop count is set to the value of the hop count from RREP message + 1
  * -  the expiry time is set to the current time plus the value of the Lifetime in the RREP message,
  * -  and the destination sequence number is the Destination Sequence Number in the RREP message.
  */
 Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
 RoutingTableEntry newEntry (/*device=*/ dev, /*dst=*/ dst, /*validSeqNo=*/ true, /*seqno=*/
rrepHeader.GetDstSeqno (),
                             /*iface=*/ m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver),
0),/*hop=*/ hop,
                             /*nextHop=*/ sender, /*lifeTime=*/ rrepHeader.GetLifeTime ());
 RoutingTableEntry toDst;
 if (m_routingTable.LookupRoute (dst, toDst))
   {
    /*
     * The existing entry is updated only in the following circumstances:
     * (i) the sequence number in the routing table is marked as invalid in route table entry.
     */
    if (!toDst.GetValidSeqNo ())
      {
       m_routingTable.Update (newEntry);
      }
    // (ii)the Destination Sequence Number in the RREP is greater than the node's copy of the destination
sequence number and the known value is valid,
    else if ((int32_t (rrepHeader.GetDstSeqno ()) - int32_t (toDst.GetSeqNo ())) > 0)
      {
       m_routingTable.Update (newEntry);
      }
    else
      {
       // (iii) the sequence numbers are the same, but the route is marked as inactive.
       if ((rrepHeader.GetDstSeqno () == toDst.GetSeqNo ()) && (toDst.GetFlag () != VALID))
         {
          m_routingTable.Update (newEntry);
         }
       // (iv)  the sequence numbers are the same, and the New Hop Count is smaller than the hop count in route
table entry.
       else if ((rrepHeader.GetDstSeqno () == toDst.GetSeqNo ()) && (hop < toDst.GetHop ()))
         {
          m_routingTable.Update (newEntry);
         }
      }
   }
 else
```

```
  {
    // The forward route for taisba destination is created if it does not already exist.
    NS_LOG_LOGIC ("add new route");
    m_routingTable.AddRoute (newEntry);
  }
// Acknowledge receipt of the RREP by sending a RREP-ACK message back
if (rrepHeader.GetAckRequired ())
  {
    SendReplyAck (sender);
    rrepHeader.SetAckRequired (true);//originally false lincy changed to true
  }
NS_LOG_LOGIC ("receiver " << receiver << " origin " << rrepHeader.GetOrigin ());
if (IsMyOwnAddress (rrepHeader.GetOrigin ()))
  {
    if (toDst.GetFlag () == IN_SEARCH)
      {
        m_routingTable.Update (newEntry);
        m_addressReqTimer[dst].Remove ();
        m_addressReqTimer.erase (dst);
      }
    m_routingTable.LookupRoute (dst, toDst);

    return;
  }

RoutingTableEntry toOrigin;
if (!m_routingTable.LookupRoute (rrepHeader.GetOrigin (), toOrigin) || toOrigin.GetFlag () == IN_SEARCH)
  {
    return; // Impossible! drop.

  }
toOrigin.SetLifeTime (std::max (ActiveRouteTimeout, toOrigin.GetLifeTime ()));
m_routingTable.Update (toOrigin);


// Update information about precursors
if (m_routingTable.LookupValidRoute (rrepHeader.GetDst (), toDst))
  {
    toDst.InsertPrecursor (toOrigin.GetNextHop ());
    m_routingTable.Update (toDst);

    RoutingTableEntry toNextHopToDst;
    m_routingTable.LookupRoute (toDst.GetNextHop (), toNextHopToDst);
    toNextHopToDst.InsertPrecursor (toOrigin.GetNextHop ());
    m_routingTable.Update (toNextHopToDst);

    toOrigin.InsertPrecursor (toDst.GetNextHop ());
    m_routingTable.Update (toOrigin);


    RoutingTableEntry toNextHopToOrigin;
    m_routingTable.LookupRoute (toOrigin.GetNextHop (), toNextHopToOrigin);
    toNextHopToOrigin.InsertPrecursor (toDst.GetNextHop ());
```

```
      m_routingTable.Update (toNextHopToOrigin);
    }

  Ptr<Packet> packet = Create<Packet> ();
  packet->AddHeader (rrepHeader);
  TypeHeader tHeader (AISBATYPE_RREP);
  packet->AddHeader (tHeader);
  Ptr<Socket> socket = FindSocketWithInterfaceAddress (toOrigin.GetInterface ());

  NS_ASSERT (socket);
  socket->SendTo (packet, 0, InetSocketAddress (toOrigin.GetNextHop (), AISBA_PORT));
}

void
RoutingProtocol::RecvReplyAck (Ipv4Address neighbor)
{
  NS_LOG_FUNCTION (taisba);
  RoutingTableEntry rt;

  if(m_routingTable.LookupRoute (neighbor, rt))
    {
      rt.m_ackTimer.Cancel ();
      rt.SetFlag (VALID);
      m_routingTable.Update (rt);

    }
}

void
RoutingProtocol::ProcessHello (RrepHeader const & rrepHeader, Ipv4Address receiver )
{
  NS_LOG_FUNCTION (taisba << "from " << rrepHeader.GetDst ());
  /*
   *  Whenever a node receives a Hello message from a neighbor, the node
   * SHOULD make sure that it has an active route to the neighbor, and
   * create one if necessary.
   */
  RoutingTableEntry toNeighbor;
  if (!m_routingTable.LookupRoute (rrepHeader.GetDst (), toNeighbor))
    {
      Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
      RoutingTableEntry newEntry (/*device=*/ dev, /*dst=*/ rrepHeader.GetDst (), /*validSeqNo=*/ true,
/*seqno=*/ rrepHeader.GetDstSeqno (),
                                  /*iface=*/ m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0),
                                  /*hop=*/ 1, /*nextHop=*/ rrepHeader.GetDst (), /*lifeTime=*/
rrepHeader.GetLifeTime ());
      m_routingTable.AddRoute (newEntry);
    }
  else
    {
      toNeighbor.SetLifeTime (std::max (Time (AllowedHelloLoss * HelloInterval), toNeighbor.GetLifeTime
()));
      toNeighbor.SetSeqNo (rrepHeader.GetDstSeqno ());
```

```
    toNeighbor.SetValidSeqNo (true);
    toNeighbor.SetFlag (VALID);
    toNeighbor.SetOutputDevice (m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver)));
    toNeighbor.SetInterface (m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0));
    toNeighbor.SetHop (1);
    toNeighbor.SetNextHop (rrepHeader.GetDst ());
    m_routingTable.Update (toNeighbor);
   }
 if (EnableHello)
   {
    m_nb.Update (rrepHeader.GetDst (), Time (AllowedHelloLoss * HelloInterval));
   }
}

void
RoutingProtocol::RecvError (Ptr<Packet> p, Ipv4Address src )//recv  route error from node with address src
{
 NS_LOG_FUNCTION (taisba << " from " << src);

 RerrHeader rerrHeader;

 p->RemoveHeader (rerrHeader);
//// PampsendHeader pampsendHeader;

              ////Lincy Starts

                //Ptr<Ipv4L3Protocol> l3K = m_ipv4->GetObject<Ipv4L3Protocol> ();
                //Ipv4InterfaceAddress ifaceLocal = l3K->GetAddress(1, 0);

              ///**lINCY  ADDED code to send PAMP when source has received RERR,ALSO ADDED
GETORIGIN,SET ORIGIN TO PACKET.CC AND H FOR AISBA
PURPOSE*****************************************************************/

     //std::cout <<  " RERR Packet Received TO node - " << ifaceLocal.GetLocal() << "\n";//lincy

          //rerrHeader.SetOrigin (ifaceLocal.GetLocal ());
           //Ipv4Address origin=rerrHeader.GetOrigin ();
        //std::cout<<" Origin  = "<<rerrHeader.GetOrigin() << "  DEST = " <<rerrHeader.GetDst()<< "\n";

         //if (IsMyOwnAddress (rerrHeader.GetOrigin())){
                                                 //std::cout<<" Calling Activate DC for " <<origin;
                                                  //ActivateDC();//Lincy
                        //}
    //Ipv4Address dst = rerrHeader.GetDst();
    //RoutingTableEntry toDst;
    //Ptr<Ipv4Route> route = toDst.GetRoute ();
      //sendPAMPSignal ( dst, route);
      ////ENDS LINCY
  std::map<Ipv4Address, uint32_t> dstWithNextHopSrc;
 std::map<Ipv4Address, uint32_t> unreachable;
 m_routingTable.GetListOfDestinationWithNextHop (src, dstWithNextHopSrc);
 std::pair<Ipv4Address, uint32_t> un;
 while (rerrHeader.RemoveUnDestination (un))
```

```
    {
     for (std::map<Ipv4Address, uint32_t>::const_iterator i =
          dstWithNextHopSrc.begin (); i != dstWithNextHopSrc.end (); ++i)
     {
      if (i->first == un.first)
        {
         unreachable.insert (un);
        }
     }
    }

  std::vector<Ipv4Address> precursors;
  for (std::map<Ipv4Address, uint32_t>::const_iterator i = unreachable.begin ();
       i != unreachable.end ();)
    {
     if (!rerrHeader.AddUnDestination (i->first, i->second))
       {
        TypeHeader typeHeader (AISBATYPE_RERR);
        Ptr<Packet> packet = Create<Packet> ();
        packet->AddHeader (rerrHeader);
        packet->AddHeader (typeHeader);
        SendRerrMessage (packet, precursors);
        rerrHeader.Clear ();
       }
     else
       {
        RoutingTableEntry toDst;
        m_routingTable.LookupRoute (i->first, toDst);
        toDst.GetPrecursors (precursors);
        ++i;
       }
    }
  if (rerrHeader.GetDestCount () != 0)
    {
     TypeHeader typeHeader (AISBATYPE_RERR);
     Ptr<Packet> packet = Create<Packet> ();
     packet->AddHeader (rerrHeader);
     packet->AddHeader (typeHeader);
     SendRerrMessage (packet, precursors);


    }
  m_routingTable.InvalidateRoutesWithDst (unreachable);
}

void
RoutingProtocol::RouteRequestTimerExpire (Ipv4Address dst)
{
  NS_LOG_LOGIC (taisba);
  RoutingTableEntry toDst;
  if (m_routingTable.LookupValidRoute (dst, toDst))
    {
     SendPacketFromQueue (dst, toDst.GetRoute ());
```

```
      NS_LOG_LOGIC ("route to " << dst << " found");
      std::cout<< "route to" << dst<< "found";//lincy
      return;
    }
  /*
   *  If a route discovery has been attempted RreqRetries times at the maximum TTL without
   *  receiving any RREP, all data packets destined for the corresponding destination SHOULD be
   *  dropped from the buffer and a Destination Unreachable message SHOULD be delivered to the application.
   */
  if (toDst.GetRreqCnt () == RreqRetries)
    {
      NS_LOG_LOGIC ("route discovery to " << dst << " has been attempted RreqRetries (" << RreqRetries <<
") times");
       std::cout << "route discovery to " << dst << " has been attempted RreqRetries (" << RreqRetries << ")
times";//lincy
      m_addressReqTimer.erase (dst);
      m_routingTable.DeleteRoute (dst);
      NS_LOG_DEBUG ("Route not found. Drop all packets with dst " << dst);
      std::cout<< "Route not found,drop all packets with dst"<<dst;//lincy
      m_queue.DropPacketWithDst (dst);
      return;
    }

  if (toDst.GetFlag () == IN_SEARCH)
    {
      NS_LOG_LOGIC ("Resend RREQ to " << dst << " ttl " << NetDiameter);
      SendRequest (dst);

      //LIncy Starts
      Ptr<Ipv4L3Protocol> l3K = m_ipv4->GetObject<Ipv4L3Protocol> ();
      Ipv4InterfaceAddress ifaceLocal = l3K->GetAddress(1,0);
      std::cout<< ifaceLocal.GetLocal() <<": resend rreq as in_Search to dst"<<dst;//lincy
    }
  else
    {
      NS_LOG_DEBUG ("Route down. Stop search. Drop packet with destination " << dst);
      m_addressReqTimer.erase (dst);
      m_routingTable.DeleteRoute (dst);
      m_queue.DropPacketWithDst (dst);
    }
}

void
RoutingProtocol::HelloTimerExpire ()
{
  NS_LOG_FUNCTION (taisba);
  Time offset = Time (Seconds (0));
  if (m_lastBcastTime > Time (Seconds (0)))
    {
      offset = Simulator::Now () - m_lastBcastTime;
      NS_LOG_DEBUG ("Hello deferred due to last bcast at:" << m_lastBcastTime);
    }
  else
```

```
    {
      SendHello ();
    }
  m_htimer.Cancel ();
  Time diff = HelloInterval - offset;
  m_htimer.Schedule (std::max (Time (Seconds (0)), diff));
  m_lastBcastTime = Time (Seconds (0));
}

void
RoutingProtocol::RreqRateLimitTimerExpire ()
{
  NS_LOG_FUNCTION (taisba);
  m_rreqCount = 0;
  m_rreqRateLimitTimer.Schedule (Seconds (1));
}

void
RoutingProtocol::RerrRateLimitTimerExpire ()
{
  NS_LOG_FUNCTION (taisba);
  m_rerrCount = 0;
  m_rerrRateLimitTimer.Schedule (Seconds (1));
}

void
RoutingProtocol::AckTimerExpire (Ipv4Address neighbor, Time blacklistTimeout)
{
  NS_LOG_FUNCTION (taisba);
  m_routingTable.MarkLinkAsUnidirectional (neighbor, blacklistTimeout);
}

void
RoutingProtocol::SendHello ()
{
  NS_LOG_FUNCTION (taisba);
  /* Broadcast a RREP with TTL = 1 with the RREP message fields set as follows:
   *   Destination IP Address         The node's IP address.
   *   Destination Sequence Number    The node's latest sequence number.
   *   Hop Count                      0
   *   Lifetime                       AllowedHelloLoss * HelloInterval
   */
  for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j = m_socketAddresses.begin (); j !=
m_socketAddresses.end (); ++j)
    {
      Ptr<Socket> socket = j->first;
      Ipv4InterfaceAddress iface = j->second;
      RrepHeader helloHeader (/*prefix size=*/ 0, /*hops=*/ 0, /*dst=*/ iface.GetLocal (), /*dst seqno=*/
m_seqNo,
                              /*origin=*/ iface.GetLocal (),/*lifetime=*/ Time (AllowedHelloLoss *
HelloInterval));
      Ptr<Packet> packet = Create<Packet> ();
      packet->AddHeader (helloHeader);
```

```
    TypeHeader tHeader (AISBATYPE_RREP);
    packet->AddHeader (tHeader);
    // Send to all-hosts broadcast if on /32 addr, subnet-directed otherwise
    Ipv4Address destination;
    if (iface.GetMask () == Ipv4Mask::GetOnes ())
      {
        destination = Ipv4Address ("255.255.255.255");
      }
    else
      {
        destination = iface.GetBroadcast ();
      }
    Time jitter = Time (MilliSeconds (m_uniformRandomVariable->GetInteger (0, 10)));
    Simulator::Schedule (jitter, &RoutingProtocol::SendTo, taisba , socket, packet, destination);
   }
}

void
RoutingProtocol::SendPacketFromQueue (Ipv4Address dst, Ptr<Ipv4Route> route)
{
 NS_LOG_FUNCTION (taisba);
 QueueEntry queueEntry;
 while (m_queue.Dequeue (dst, queueEntry))
   {
    DeferredRouteOutputTag tag;
    //Lincy Starts
    //Ptr<Ipv4L3Protocol> l3K = m_ipv4->GetObject<Ipv4L3Protocol> ();
    //Ipv4InterfaceAddress ifaceLocal = l3K->GetAddress(1,0);
    //std::cout<< ifaceLocal.GetLocal() <<": send packet from queue\n";//lincy
    //Lincy Ends
    Ptr<Packet> p = ConstCast<Packet> (queueEntry.GetPacket ());
    if (p->RemovePacketTag (tag) &&
        tag.GetInterface() != -1 &&
        tag.GetInterface() != m_ipv4->GetInterfaceForDevice (route->GetOutputDevice ()))
      {
        NS_LOG_DEBUG ("Output device doesn't match. Dropped.");
        return;
      }
    UnicastForwardCallback ucb = queueEntry.GetUnicastForwardCallback ();
    Ipv4Header header = queueEntry.GetIpv4Header ();
    header.SetSource (route->GetSource ());
    header.SetTtl (header.GetTtl () + 1); // compensate extra TTL decrement by fake loopback routing
    ucb (route, p, header);
   }
}

void
RoutingProtocol::SendRerrWhenBreaksLinkToNextHop (Ipv4Address nextHop)
{
 NS_LOG_FUNCTION (taisba << nextHop);
 //std::cout<<nextHop<<"NEXTHOP";
 RerrHeader rerrHeader;
 std::vector<Ipv4Address> precursors;
```

```
  std::map<Ipv4Address, uint32_t> unreachable;

  RoutingTableEntry toNextHop;
  if (!m_routingTable.LookupRoute (nextHop, toNextHop))
    return;
  toNextHop.GetPrecursors (precursors);
  rerrHeader.AddUnDestination (nextHop, toNextHop.GetSeqNo ());
  m_routingTable.GetListOfDestinationWithNextHop (nextHop, unreachable);
  for (std::map<Ipv4Address, uint32_t>::const_iterator i = unreachable.begin (); i
       != unreachable.end ();)
    {
      if (!rerrHeader.AddUnDestination (i->first, i->second))
        {
          NS_LOG_LOGIC ("Send RERR message with maximum size.");
          TypeHeader typeHeader (AISBATYPE_RERR);
          Ptr<Packet> packet = Create<Packet> ();
          packet->AddHeader (rerrHeader);
          packet->AddHeader (typeHeader);
          SendRerrMessage (packet, precursors);
          rerrHeader.Clear ();
        }
      else
        {
          RoutingTableEntry toDst;
          m_routingTable.LookupRoute (i->first, toDst);
          toDst.GetPrecursors (precursors);
          ++i;
        }
    }
  if (rerrHeader.GetDestCount () != 0)
    {
      TypeHeader typeHeader (AISBATYPE_RERR);
      Ptr<Packet> packet = Create<Packet> ();
      packet->AddHeader (rerrHeader);
      packet->AddHeader (typeHeader);
      SendRerrMessage (packet, precursors);
    }
  unreachable.insert (std::make_pair (nextHop, toNextHop.GetSeqNo ()));
  m_routingTable.InvalidateRoutesWithDst (unreachable);
}

void
RoutingProtocol::SendRerrWhenNoRouteToForward (Ipv4Address dst,
                                               uint32_t dstSeqNo, Ipv4Address origin)
{
  NS_LOG_FUNCTION (taisba);
  // A node SHOULD NOT originate more than RERR_RATELIMIT RERR messages per second.
  if (m_rerrCount == RerrRateLimit)
    {
      // Just make sure that the RerrRateLimit timer is running and will expire
      NS_ASSERT (m_rerrRateLimitTimer.IsRunning ());
      // discard the packet and return
```

```
    NS_LOG_LOGIC ("RerrRateLimit reached at " << Simulator::Now ().GetSeconds () << " with timer delay
left "
                                  << m_rerrRateLimitTimer.GetDelayLeft ().GetSeconds ()
                                  << "; suppressing RERR");
    return;
   }
  RerrHeader rerrHeader;
  rerrHeader.AddUnDestination (dst, dstSeqNo);
  RoutingTableEntry toOrigin;
  Ptr<Packet> packet = Create<Packet> ();
  packet->AddHeader (rerrHeader);
  packet->AddHeader (TypeHeader (AISBATYPE_RERR));
  if (m_routingTable.LookupValidRoute (origin, toOrigin))
    {
                          //Lincy Starts
                          Ptr<Ipv4L3Protocol> l3K = m_ipv4->GetObject<Ipv4L3Protocol> ();
     Ipv4InterfaceAddress ifaceLocal = l3K->GetAddress(1,0);
                          //std::cout<< ifaceLocal.GetLocal() << ": " <<origin <<"hey i found you-origin
\n";//lincy
                          //sendPAMPSignal();
                          //Lincy Ends
     Ptr<Socket> socket = FindSocketWithInterfaceAddress (
        toOrigin.GetInterface ());
     NS_ASSERT (socket);
     NS_LOG_LOGIC ("Unicast RERR to the source of the data transmission");
    std::cout<< ifaceLocal.GetLocal() << ": " << "Unicast RERR to "<<origin <<"\n";//lincy
     std::cout<<origin<<"initiator of RREQ"<<"\n";//lincy
      RerrHeader rerrHeader;//lincy


              /**lINCY  ADDED code to send PAMP when source has received RERR,ALSO ADDED
GETORIGIN,SET ORIGIN TO PACKET.CC AND H FOR TAISBA
PURPOSE**********************************************************/
                    // rerrHeader.SetOrigin (ifaceLocal.GetLocal ());
          //  Ipv4Address origin=rerrHeader.GetOrigin ();


         // if (IsMyOwnAddress (origin)){
                                                 std::cout<<" Calling Activate DC for " <<origin;
                                                  ActivateDC(origin);//Lincy
                     //  }

        //SendRerrMessage (packet, precursors);
      //Ipv4Address dst = rerrHeader.GetDst();
      RoutingTableEntry toOrigin;
      Ptr<Ipv4Route> route = toOrigin.GetRoute ();
        sendPAMPSignal ( origin, route);

       //ENDS LINCY
      //if (IsMyOwnAddress (origin)) //lincy
      //{ //(m_routingTable.LookupValidRoute (origin, toOrigin));
                                      ////sendPAMPSignal();
                                      //std::cout<<"JKJKJKJKJJJKJKKJKJKJKJKJKJKJK";
```

```
                                        //return;} //lincy
    socket->SendTo (packet, 0, InetSocketAddress (toOrigin.GetNextHop (), AISBA_PORT));
   }
  else
   {
    for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator i =
          m_socketAddresses.begin (); i != m_socketAddresses.end (); ++i)
      {
       Ptr<Socket> socket = i->first;
       Ipv4InterfaceAddress iface = i->second;
       NS_ASSERT (socket);
       NS_LOG_LOGIC ("Broadcast RERR message from interface " << iface.GetLocal ());
       // Send to all-hosts broadcast if on /32 addr, subnet-directed otherwise
       Ipv4Address destination;
       if (iface.GetMask () == Ipv4Mask::GetOnes ())
         {
          destination = Ipv4Address ("255.255.255.255");
         }
       else
         {
          destination = iface.GetBroadcast ();
         }
       socket->SendTo (packet->Copy (), 0, InetSocketAddress (destination, AISBA_PORT));
      }
   }
}

void
RoutingProtocol::SendRerrMessage (Ptr<Packet> packet, std::vector<Ipv4Address> precursors)
{
 NS_LOG_FUNCTION (taisba);


 if (precursors.empty ())
   {
    NS_LOG_LOGIC ("No precursors");
    return;
   }
 // A node SHOULD NOT originate more than RERR_RATELIMIT RERR messages per second.
 if (m_rerrCount == RerrRateLimit)
   {
    // Just make sure that the RerrRateLimit timer is running and will expire
    NS_ASSERT (m_rerrRateLimitTimer.IsRunning ());
    // discard the packet and return
    NS_LOG_LOGIC ("RerrRateLimit reached at " << Simulator::Now ().GetSeconds () << " with timer delay
left "
                                  << m_rerrRateLimitTimer.GetDelayLeft ().GetSeconds ()
                                  << "; suppressing RERR");
                                  std::cout <<"RerrRateLimit reached at " << Simulator::Now ().GetSeconds () << "
with timer delay left "
                                  << m_rerrRateLimitTimer.GetDelayLeft ().GetSeconds ()
                                  << "; suppressing RERR";//lincy
    return;
```

```
    }
  // If there is only one precursor, RERR SHOULD be unicast toward that precursor
  if (precursors.size () == 1)
    {
      RoutingTableEntry toPrecursor;
      if (m_routingTable.LookupValidRoute (precursors.front (), toPrecursor))
        {
          Ptr<Socket> socket = FindSocketWithInterfaceAddress (toPrecursor.GetInterface ());
          NS_ASSERT (socket);
          NS_LOG_LOGIC ("one precursor => unicast RERR to " << toPrecursor.GetDestination () << " from " <<
toPrecursor.GetInterface ().GetLocal ());
          Simulator::Schedule (Time (MilliSeconds (m_uniformRandomVariable->GetInteger (0, 10))),
&RoutingProtocol::SendTo, taisba, socket, packet, precursors.front ());
          m_rerrCount++;
        }
      return;
    }

  //  Should only transmit RERR on those interfaces which have precursor nodes for the broken route
  std::vector<Ipv4InterfaceAddress> ifaces;
  RoutingTableEntry toPrecursor;
  for (std::vector<Ipv4Address>::const_iterator i = precursors.begin (); i != precursors.end (); ++i)
    {
      if (m_routingTable.LookupValidRoute (*i, toPrecursor) &&
          std::find (ifaces.begin (), ifaces.end (), toPrecursor.GetInterface ()) == ifaces.end ())
        {
          ifaces.push_back (toPrecursor.GetInterface ());
        }
    }

  for (std::vector<Ipv4InterfaceAddress>::const_iterator i = ifaces.begin (); i != ifaces.end (); ++i)
    {
      Ptr<Socket> socket = FindSocketWithInterfaceAddress (*i);
      NS_ASSERT (socket);
      NS_LOG_LOGIC ("Broadcast RERR message from interface " << i->GetLocal ());
      // std::cout << "Broadcast RERR message from interface " << i->GetLocal () << std::endl;
      // Send to all-hosts broadcast if on /32 addr, subnet-directed otherwise
      Ptr<Packet> p = packet->Copy ();
      Ipv4Address destination;
      if (i->GetMask () == Ipv4Mask::GetOnes ())
        {
          destination = Ipv4Address ("255.255.255.255");
        }
      else
        {
          destination = i->GetBroadcast ();
        }
      Simulator::Schedule (Time (MilliSeconds (m_uniformRandomVariable->GetInteger (0, 10))),
&RoutingProtocol::SendTo, taisba, socket, p, destination);
    }
}

Ptr<Socket>
```

```
RoutingProtocol::FindSocketWithInterfaceAddress (Ipv4InterfaceAddress addr ) const
{
 NS_LOG_FUNCTION (taisba << addr);
 for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
       m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
   {
     Ptr<Socket> socket = j->first;
     Ipv4InterfaceAddress iface = j->second;
     if (iface == addr)
       return socket;
   }
 Ptr<Socket> socket;
 return socket;
}

Ptr<Socket>
RoutingProtocol::FindSubnetBroadcastSocketWithInterfaceAddress (Ipv4InterfaceAddress addr ) const
{
 NS_LOG_FUNCTION (taisba << addr);
 for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
       m_socketSubnetBroadcastAddresses.begin (); j != m_socketSubnetBroadcastAddresses.end (); ++j)
   {
     Ptr<Socket> socket = j->first;
     Ipv4InterfaceAddress iface = j->second;
     if (iface == addr)
       return socket;
   }
 Ptr<Socket> socket;
 return socket;
}

void
RoutingProtocol::DoInitialize (void)
{
 NS_LOG_FUNCTION (taisba);
 uint32_t startTime;
 if (EnableHello)
   {
     m_htimer.SetFunction (&RoutingProtocol::HelloTimerExpire, taisba);
     startTime = m_uniformRandomVariable->GetInteger (0, 100);
     NS_LOG_DEBUG ("Starting at time " << startTime << "ms");
     m_htimer.Schedule (MilliSeconds (startTime));
   }
 Ipv4RoutingProtocol::DoInitialize ();
}

// Method added by Lincy

//void RoutingProtocol::sendPAMPSignal(){
  //RoutingTableEntry _oRoutingTableEntry;
  //Ptr<Ipv4L3Protocol> l3K = m_ipv4->GetObject<Ipv4L3Protocol> ();
  //Ipv4InterfaceAddress ifaceLocal = l3K->GetAddress(1,0);
  //std::cout << ifaceLocal.GetLocal() << ": " << "Talking from sendPAMPSignal Method\n";
```

```
//}
/////////////////SENDPAMPSIGANL///LINCY
//void
//RoutingProtocol::sendPAMPSignal (Ipv4Address dst, Ipv4Address origin)
//{
  //std::cout<<"ARE YOU REACHABLE"<<dst<<"\n";//LINCY
//}
void
RoutingProtocol::sendPAMPSignal (Ipv4Address dst, Ptr<Ipv4Route> route)
{
  //NS_LOG_FUNCTION (taisba);
  //QueueEntry queueEntry;
  //while (m_queue.Dequeue (dst, queueEntry))
    //{
      //DeferredRouteOutputTag tag;
      //Lincy Starts
     // Ptr<Ipv4L3Protocol> l3K = m_ipv4->GetObject<Ipv4L3Protocol> ();
//Ipv4InterfaceAddress ifaceLocal = l3K->GetAddress(1,0);
     // std::cout<< ifaceLocal.GetLocal() <<": send PAMP\n";//lincy
     RerrHeader rerrHeader;
                //Ptr<Ipv4L3Protocol> l3K = m_ipv4->GetObject<Ipv4L3Protocol> ();
 // Ipv4InterfaceAddress ifaceLocal = l3K->GetAddress(1,0);
//  rerrHeader.SetOrigin (ifaceLocal.GetLocal ());
rerrHeader.SetOrigin(dst);
                Ipv4Address ac =rerrHeader.GetOrigin();
                std::cout<<"sendpamp" << ac;
      //Lincy Ends

    //  RerrHeader rerrHeader;
      // rerrHeader.SetOrigin (ifaceLocal.GetLocal ());
           //  Ipv4Address origin=rerrHeader.GetOrigin ();

           //if (IsMyOwnAddress (origin)){
                                                    //ActivateDC();//Lincy
                        //}
    //  Ipv4Address dst = rerrHeader.GetDst();
       //Ipv4Address origin = rerrheader.GetOrigin();
 // m_routingTable.Purge ();
  //RoutingTableEntry toDst;

        //  Ptr<Ipv4Route> routes = toDst.GetRoute ();

        //  RoutingTableEntry toOrigin;
        //  m_routingTable.LookupRoute (origin, toOrigin);
        //  UpdateRouteLifeTime (toOrigin.GetNextHop (), ActiveRouteTimeout);

         // m_nb.Update (route->GetGateway (), ActiveRouteTimeout);
        //  m_nb.Update (toOrigin.GetNextHop (), ActiveRouteTimeout);

    //PampsendHeader pampsendHeader;
      //Ptr<Packet> packet = Create<Packet> ();
//packet->AddHeader (rerrHeader);//pampsendHeader rerrHeader
     // TypeHeader tHeader (AISBATYPE_RERR);//AISBATYPE_PAMPSEND
```

```
   //  packet->AddHeader (tHeader);
    // Ptr<Packet> p = ConstCast<Packet> (queueEntry.GetPacket ());
     //if (p->RemovePacketTag (tag) &&
        //tag.GetInterface() != -1 &&
        //tag.GetInterface() != m_ipv4->GetInterfaceForDevice (route->GetOutputDevice ()))
       //{
        //NS_LOG_DEBUG ("Output device doesn't match. Dropped.");
        //return;
       //}
     //UnicastForwardCallback ucb = queueEntry.GetUnicastForwardCallback ();
     //Ipv4Header header = queueEntry.GetIpv4Header ();
     //header.SetSource (route->GetSource ());
     //header.SetTtl (header.GetTtl () + 1); // compensate extra TTL decrement by fake loopback routing
     //ucb (route, p, header);
   //}
}

void
RoutingProtocol::ackPAMPSignal (RreqHeader const & rreqHeader, RoutingTableEntry const & toOrigin)
{
                std::cout<<"I AM REACHABLE"<<"\n";//LINCY
                //bool ackPAMPSignalrecv;//doubt
}
//Method added bt Lincy
void RoutingProtocol::ActivateDC(Ipv4Address origins){

                RerrHeader rerrHeader;
                //Ptr<Ipv4L3Protocol> l3K = m_ipv4->GetObject<Ipv4L3Protocol> ();
 // Ipv4InterfaceAddress ifaceLocal = l3K->GetAddress(1,0);
//  rerrHeader.SetOrigin (ifaceLocal.GetLocal ());
rerrHeader.SetOrigin(origins);
                Ipv4Address dc =rerrHeader.GetOrigin();
                std::cout<<"ActivateDC" << dc;
  //RoutingTableEntry _oRoutingTableEntry;
 // Ptr<Ipv4L3Protocol> l3K = m_ipv4->GetObject<Ipv4L3Protocol> ();
 //// Ipv4InterfaceAddress ifaceLocal = l3K->GetAddress(1,0);
 // std::cout << ifaceLocal.GetLocal() << ": " << "Activate DC\n";
}

} //namespace aisba
} //namespace ns3
```

Appendix 3 Value of Node nearness facor(K)

Hop count (F) ln(F)	K	ln(K)	Pcom=ln(K)/ln(F) (for F=14)
2 .693	1	0	0
4 1.3	2	.693	0.26
6 1.7	3	1.09	0.41
8 2.0	4	1.38	0.53
10 2.3	5	1.60	0.61
12 2.4	6	1.79	0.68
14 2.6	9	2.19	0.84

As can be observed ,when the value of K gets higher this gives a misleading information regarding Pcom (Probability of node nearness/communication).For example, if we consider a higher value of hop count ,it is most likely that the node is far away and the probability of communication is very low.Thereby choosing higher value of K ,along with a higher hop count , shows that the probability of communication is high which is not correct therefore the ideal value of K is chosen to be 2.The same can be inferred from the table above when K is 9and hop count is 14 the value of Pcom is 0.84 which is misleading.Therefore the value of K should be chosen in accordance with the Probability of node nearness/communication and signal the correct information about the node nearness.