MAC LAYER MISBEHAVIOR EFFECTIVENESS AND COLLECTIVE AGGRESSIVE REACTION APPROACH

A Thesis by

VamshiKrishna Reddy Giri

Bachelor of Technology, Jawaharlal Nehru Technological University, 2008

Submitted to the Department of Electrical Engineering and Computer Science
and the faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Master of Science

December 2010

MAC LAYER MISBEHAVIOR EFFECTIVENESS AND COLLECTIVE AGGRESSIVE REACTION APPROACH

The following faculty members have examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science with a major in Electrical Engineering.

_____
Neeraj Jaggi, Committee Chair

_____
Vinod Namboodiri, Commitee Member

_____
Hamid M. Lankarani, Commitee Member

DEDICATION

To Nikhil Marrapu, Sreenivas Madakasira, Dr. Neeraj Jaggi, and my family

ACKNOWLEDGEMENTS

It gives me immense pleasure to acknowledge all the people who have been very helpful to me throughout my Master's program. My mother, Mrs. Suvarna Pratap Reddy Giri, is the most important person in my life, Without her support and love I would not be in the place where I am today. I thank her for always believing in me.

I would like to thank my research advisor, Dr. Neeraj Jaggi, for his constant support and guidance. Working under him was a priceless learning experience, which helped me understand the importance of precision and presentation in research.

Apart from academics, he has also been a motivational figure for me, and has truly influenced me in many ways, for which I will always be grateful. I am heartily thankful to Umesh Marappa Reddy, Abhinav Mallepilla, Anudeep, Lakshman, Nishanth, and Vinay Baba for making my stay at Wichita State University wonderful.

I thank my research associates Nikhil Marrapu,and Sreenivas Madakasira for all their help and support. I am indebted to them for taking care of me like their brother and making me feel at home always. They have been not just friends but family to me.

My sincere regards to Dr. Vinod Namboodiri and Dr. Hamid Lankarani for their time and valuable suggestions.

ABSTRACT

Current wireless MAC protocols are designed to provide an equal share of throughput to all nodes in the network. However, the presence of misbehaving nodes (selfish nodes that deviate from standard protocol behavior in order to obtain higher bandwidth) poses severe threats to the fairness aspects of MAC protocols. In this thesis, investigation of various types of MAC layer misbehaviors is done, and their effectiveness is evaluated in terms of their impact on important performance aspects including throughput, and fairness to other users. Observations obtained from the simulation of misbehaviors show that the effects of misbehavior are prominent only when the network traffic is sufficiently large and the extent of misbehavior is reasonably aggressive. In addition, it is also observed that the performance gains achieved using misbehavior exhibit diminishing returns with respect to its aggressiveness, for all types of misbehaviors considered. Crucial common characteristics among such misbehaviors are identified, and these learnings are used to design an effective measure to react towards such misbehaviors.

Employing two of the most effective misbehaviors, it is shown that collective aggressiveness of non-selfish nodes is a possible strategy to react towards selfish misbehavior. Particularly, a dynamic collective aggressive reaction approach is demonstrated to ensure fairness in the network, however at the expense of overall network throughput degradation. In addition, the proposed adaptive reaction strategy provides the necessary disincentive to prevent selfish misbehavior in the network.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (continued)

Figure Page

LIST OF FIGURES (continued)

Figure Page

LIST OF ABBREVIATIONS / NOMENCLATURE

ACK	Acknowledgement
AP	Access Point
b	Fixed Backoff value
BEB	Binary Exponential Backoff
BEB Pkt Threshold	BEB Packet Threshold
CTS	Clear to Send
CW	Contention Window
CW_{fix}	Fixed Contention Window
CW_{max}	Maximum Contention window
CW_{min}	Minimum Contention Window
DCF	Distributed Coordination Function
DIFS	Distributed Inter Frame Space
MAC	Medium Access Control
NS-2	Network Simulator 2
PCF	Point Coordination Function
Rea Pkt Threshold	Reaction Packet Threshold
RTS	Request to Send
SIFS	Short Inter Frame Space
T_g	Genuine Throughput
T_m	Throughput of a Misbehaving Node

LIST OF SYMBOLS

α Misbehavior Parameter used in α-Misbehavior

β Misbehavior Parameter used in β-Misbehavior

CHAPTER 1

INTRODUCTION

MAC protocols are intended to provide a fair access to the wireless channel for all users in the wireless LAN. Such fairness serves as a backbone to the design of more sophisticated service differentiation mechanisms in order to provide quality of service in the network. If a node manages to bypass the standard MAC protocols, then it gets an increased share of access to the medium, thus resulting in fairness issues in the network. Due to such misbehaviors at the MAC layer, nodes gain an increased throughput share of the network. The core idea of this thesis is to present a reaction approach against the Mac layer misbehavior. The primary focus of the reaction approach is to prevent such misbehavior and to ensure fairness among the nodes in a wireless network.

1.1 Binary Exponential Backoff in IEEE 802.11 MAC Protocol

The IEEE 802.11 protocol [7] employs binary exponential backoff (BEB) scheme to introduce randomness in channel access, and avoids collision by relying on the nodes to double their contention window (CW) upon collision. BEB specifies that every node must choose a random back off value from a contention window with uniform probability distribution before accessing the wireless channel. Every node must wait for the backoff value amount of time before accessing the channel. This randomness in choice of backoff value guarantees fairness in the network. Figure 1.1 [12], depicts the cases in which the BEB algorithm is executed.

The cases can be summarized as follows:

- When the node senses the channel for the first transmission and finds it busy.
- After each successful data transmission.
- After each failed transmission attempt.

The only exception is the case where a node senses channel idle (IDLE) for more than the distributed inter frame space (DIFS) amount of time, and the node has a new packet to send. In

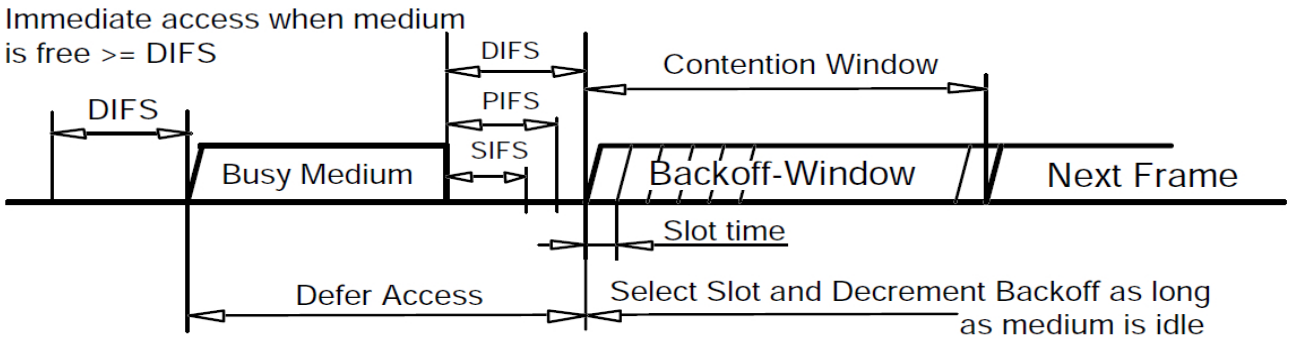

Figure 1.1 – Binary exponential backoff [12]

this case, the node transmits immediately and does not wait for channel access. The BEB algorithm can be explained as follows :

- A node that has data to transmit chooses a backoff value *b* uniformly from interval [0 .. . CW-1], where CW denotes the current contention window size at the node.
- If the wireless channel is sensed to be IDLE, then the node waits for *b* time slots before accessing the channel. This is done by decrementing *b* by 1 for every time slot it finds the channel IDLE.
- If the wireless channel is sensed busy, then the node freezes its backoff *b* until the channel is sensed idle again and continues counting down thereafter.
- The backoff value is always chosen in the range $\{0, \text{CW-1}\}$.
- The initial value of contention window from which backoff is chosen is CW = CW_{min}. The value of CW_{min} according to IEEE standard is 32.
- After waiting for backoff amount of time, the node transmits the data onto the channel.
- If the transmission is successful, then the node resets its CW to CW_{min}.

- If transmission is unsuccessful, then node sets CW to min$\{2$ * CW, $\text{CW}_{max}\}$. According to IEEE standards the value of CW_{max} is 1024.

The algorithm is robust and ensures fairness in the system even in saturated traffic conditions. However, any alteration to the parameters in this algorithm may result in an increased share of channel bandwidth for the individual node.

1.2 MAC Layer Misbehavior

Misbehavior in a network can be described as the noncompliance of peer nodes in a network to follow the standard protocols. However, increased programmability of network devices lately has led to the possibility of individual users modifying their own protocol behavior in order to achieve higher bandwidth. Such misbehaviors at the MAC layer adversely affect the performance of the network in terms of overall throughput and fairness, and are quite intense when distributed coordination function (DCF) mode of 802.11 operation is in use[1]. Misbehaviors can be due to the alteration of various parameters in the BEB algorithm. Typical parameters that can be easily configured are the backoff counter, DIFS and short inter frame space (SIFS) intervals, network allocation vector (NAV), contention window etc.

Misbehaviors can be classified into two types based on the objectives of the misbehaving node: greedy misbehaviors or malicious misbehaviors. A user can employ multiple strategies in order to achieve the objectives. The objective of the greedy misbehavior would be to simply obtain an increased share of bandwidth. Greedy users employ mildly aggressive cheating with respect to backoff rules [8], [9], [10] or the choice of contention window [11] in order to increase their own throughput share at the expense of other genuine nodes. The most common ways to achieve such misbehaviors include :

[1]In 802.11 DCF mode, all nodes contend for the channel using BEB, whereas the other mode, PCF, uses an access point-based architecture.

- Choice of a small fixed value as backoff counter. Here the node always waits for very a small amount of time before accessing the IDLE wireless channel.
- Choice of a very small contention window, always resulting in a smaller backoff value.
- Choice of small DIFS, SIFS etc.

The objective of the malicious misbehavior would be to disrupt the normal network behavior using jamming, denial-of-service attacks, or non-cooperation with respect to data forwarding [5].

1.3 Contributions of This Thesis

This thesis work considers greedy (selfish) misbehaviors pertaining to the 802.11 DCF, obtained by the modifications made to the BEB algorithm. The contributions of this thesis can be briefly summarized as follows:

- Five types of misbehaviors that can be achieved exploiting the BEB algorithm are defined.
- These types of misbehaviors are simulated in OPNET simulator and in NETWORK SIMULATOR (NS-2).
- Study of their effectiveness, and impact on network performance under varying traffic load scenarios is performed.
- Common characteristics are observed among different types of misbehaviors and are noted. Comments are made on those cheating strategies that may be employed by a smart cheater to avoid detection while simultaneously achieving a higher share of the throughput.
- Scenarios where it is (or is not) appropriate to trigger a reaction response are identified.
- Fairness aspects are studied, with and without the presence of misbehavior, and when the reaction strategy is applied.

- A collective aggressive misbehavior response by genuine nodes is proposed as a strategy to react towards misbehavior, and it is demonstrated that such an approach guarantees fairness in the network.

- A throughput-based distributed detection scheme is demonstrated to identify potential misbehaviors, which could be used to employ the reaction strategy in a distributed manner.

1.4 Thesis Structure

The rest of this thesis is organized as follows: Chapter 2 presents the literary overview of the work done so far in this area of research. Chapter 3 presents the definitions of various types of misbehaviors that are introduced to study the effects of misbehaviors on network performance. Chapter 4 introduces effectiveness as a performance measure to compare different types of misbehaviors. It also provides the analysis of the effectiveness plots obtained for various traffic loads and comments on their behavior. A new reaction strategy is presented in chapter 5. Fairness aspects that confirm the intuition behind the reaction strategy are also discussed. A distributed dynamic and adaptive reaction scheme is presented and the thesis concludes with future research work in Chapter 6. Part of this thesis has been published in a conference[13].

CHAPTER 2

RELATED WORK

2.1 Previous Work

Misbehavior at the MAC layer is considered in many contexts, and numerous detection methodologies and reaction schemes have been proposed. A detection scheme based on observed backoff intervals chosen by other nodes and employing a sequential probability ratio test have been proposed in [1]. Other schemes employ throughput degradation [2] or access point-based adaptive mechanism [3] to detect presence of misbehavior. Kyassanur and Vaidya in [4] introduce the concept of receiver-assigned backoff, and Guang et al. [5] propose modifications to the BEB algorithm in order to facilitate easy detection and penalization of the misbehaving sender.

Radoavac et al. [1] and Kyasanur and Vaidya [4] discuss schemes for detection of misbehavior caused by the choice of small backoff counter values. Radoavac et al. of [1] present a minimax robust approach to detect the misbehavior based on the sequence of backoff values observed by a node. Each node observes does the following:

- The sequence of backoff values chosen by other nodes.
- Applies a Wald's sequential probability ratio test(SPRT) to these observations.
- Has the SPRT give a ratio S_k as the output.

The value of S_k determines the selection of one of the two predetermined hypotheses (misbehaving or genuine).

Unlike the standard method of the sender choosing the backoff values, Kyasanur and Vaidya [4] propose receiver-assigned backoff values to detect the presence of misbehaviors. The receiver selects the backoff value that is to be used by the sender for the subsequent transmissions and sends it to the sender in its clear to send (CTS) or acknowledgement (ACK) packet. The receiver then observes the number of idle slots for which the sender must wait before the transmission, and if

that happens to be less than the assigned backoff, then it flags the sender as misbehaving. Based on the deviation of the sender from assigned backoff the receiver penalizes the misbehaving sender by increasing the backoff values assigned thereafter. After repeated misbehavior by the sender, the receiver assigns a percentage of misbehavior (PM) to the sender. A zero value of PM indicates that the sender counts down the backoff value to 0 without any misbehavior and a value of 100 indicates that sender ignores backoff and transmits immediately. The receiver takes this into account before penalizing the misbehavior.

Lee et al. [6] model exponential backoff at MAC layer as a non-cooperative game model. Raya et al. [3] present DOMINO, an adaptive system for detecting misbehaviors deployed at an access point(AP). The system conducts a series of tests checking for different possible misbehaviors in the network. Every test consists of a deviation estimation component(DEC) and a anomaly detection component (ADC). The results of these test are then sent to a decision making component (DMC). At DMC the results are aggregated using a function and a weighted result is generated. The result is then used to decide upon the misbehavior.

Guang et al. [5] discuss the transmission time out of MAC frames due to the selfish behavior either at the receiver or the sender. They propose a novel detection and reaction system detection and reaction to mac layer misbehavior (DREAM). to detect and punish the misbehaving peer. The system provides a two stage scheme (first and second reaction schemes) to achieve the detection and reaction. Each of genuine nodes are installed with DREAM system. The first reaction scheme tries to identify the potential misbehaving nodes using a bad credit value. Any node whose bad credit reaches a credit threshold is identified a suspect node (SN). The second reaction procedure assigns a trust level for the SN to classify SN as either a trusted node (TN) or a untrusted node (UN) :

- Every time an SN misbehaves its trust level is reduced.
- If this trust level falls below a trust threshold then the node is marked as a UN.
- If a node is marked as a UN then subsequent methods of punishments methods are invoked.

The punishment scheme involves propagating the trust level of the misbehaving to upper layers and preventing traffic via it. This kind of penalization in some way is denial of service by the network to the misbehaving nodes.

2.2 Motivation

Most of the research considering MAC-layer misbehavior has been done for some known type of misbehavior. Most reaction schemes employed by genuine nodes attempt to penalize [4] or isolate, Guang et al.[3] isolate the selfish node.[5] suggests that isolation of misbehaving nodes is not the best reaction, as it affects performance at the network and the higher layers, as greater number of nodes get isolated.

This thesis, study of different types of misbehaviors is done in an attempt to understand the common characteristics among them, which could be employed to detect misbehaviors and trigger a reaction response. The overall objective of the reaction response is to make it disadvantageous for any user to deviate from standard protocol behavior. This work proposes a reaction scheme which not only guarantees fairness, but also provides the needed disincentive to the selfish user in order to prevent misbehavior, without isolating the selfish node.

2.3 Summary

This chapter, discusses the previous work in the area of misbehavior detection and reaction at MAC-layer. The key aspects that are presented in this thesis are briefly discussed against the intuitions drawn from earlier research.

CHAPTER 3

MAC-LAYER MISBEHAVIOR CLASSIFICATION

3.1 Introduction

Various types of misbehaviors can be generated to modify the BEB algorithm.This chapter focuses on defining various ways in which a selfish user can misbehave. The behavior of the normal node following the standard BEB algorithm can be summarized as follows: A node that has data to transmit chooses a backoff interval b uniformly at random from the interval [0 . . .CW-1], where CW denotes the node's current contention window size. The node waits for b time slots before accessing the channel. However, if the channel is sensed busy during this time, the node freezes its backoff until the channel is sensed idle again and continues counting down thereafter. The initial contention window size equals CW_{min}. Upon successful transmission, the node resets its CW to CW_{min}. However, upon unsuccessful transmission (eg.,due to collision), the node sets its CW to be min$\{2 \times$ CW, $CW_{max}\}$. Standard choices for the above constants are given by CW_{min} = 32 and CW_{max} = 1024.

3.2 Misbehavior Types

Five types of misbehaviors associated with modifying the 802.11 BEB algorithm are identified.

3.2.1 α-misbehavior

- Instead of choosing the backoff *b* uniformly at random from the interval [0 . . .CW-1], the selfish node chooses b uniformly at random from the interval [0 . . .α(CW-1)].
- The value of α lies in the range [0 . . .1].

- The node ends up choosing a smaller backoff interval than is typical thus, increasing its chances of accessing the channel next.

3.2.2 β-misbehavior

- Upon unsuccessful transmission, the node, instead of setting its CW to be min$\{2 \times$ CW, CW$_{max}\}$, sets its contention window as CW = max$\{$CW$_{min}$, min$\{\beta \times$ CW, CW$_{max}$ $\}\}$.
- The value of β lies in the range [0 . . .2].
- This results in the node choosing smaller backoff interval than expected.
- Also, the node sets CW$_{min}$ = min$\{32, \beta \times 32\}$ to appropriately distinguish between the scenarios when $\beta < 1$.

3.2.3 Fixed Maximum Contention Window (CW$_{max}$)

- The typical value of CW$_{max}$ employed by genuine nodes equals 1,024. However, the selfish node employing CW$_{max}$-misbehavior sets its maximum contention window to be a value smaller than 1024.
- Thus, the node ends up choosing smaller backoff intervals than other genuine nodes, particularly at higher traffic loads when the number of collisions in the network increases.
- Also, the node sets CW$_{min}$ = min$\{32,$ CW$_{max}$ $\}$, to address those scenarios where the chosen maximum contention window is smaller than CW$_{min}$, which equals 32 from standard 802.11 [7].

3.2.4 Fixed Contention Window (CW_{fix})-misbehavior

- The selfish node sets its contention window to a small, fixed size CW_{fix}, and always chooses its backoff interval uniformly at random from the interval [0 . . .CW_{fix}-1].
- Thus, the node avoids BEB, resulting in a higher share of bandwidth achieved.
- The smaller the value of CW_{fix}, more aggressive is the misbehavior.[2]

3.2.5 Deterministic Backoff (db)-misbehavior

- Instead of choosing a backoff value uniformly distributed from a window, the node chooses a deterministic, constant backoff interval *b*, irrespective of size of the current contention window.
- For instance, the node could always choose a very small backoff (say 2), irrespective of multiple failed transmission attempts, thus trying to gain preference over other genuine nodes in terms of channel access.

In all of the above described misbehaviors, there is always increased contention in the network, thus affecting overall LAN throughput. But the misbehaving node still obtains the desired increased throughput. The effectiveness of these misbehaviors at various traffic loads are evaluated in Chapter 4. Employing one or more of these misbehaviors together, multiple hybrid misbehaviors can be generated. An interesting question that arises is whether employing the hybrid misbehaviors yield better results for the misbehaving node. This question is explored in Chapter 4.

[2]In all different types of misbehavior, the selfish node could vary the level or aggressiveness of its misbehavior by appropriately choosing of the involved parameters. For instance, the smaller the value of CW_{fix}, the more aggressive the CW_{fix}-misbehavior. In addition, a node may employ a hybrid strategy, which is a combination of two or more of the above.

3.3 Summary

This chapter classifies five types of misbehaviors based on modifications to BEB algorithm. Varying the aggressiveness of misbehavior by varying the parameters involved is also explained.

CHAPTER 4

MISBEHAVIOR EFFECTIVENESS CHARACTERIZATION

4.1 Misbehavior Effectiveness

The impact of each of the misbehavior's mentioned in chapter 3 on the performance of a wireless network can be better studied if this impact is measured appropriately. In order to measure the effectiveness of a misbehavior, the percentage of increase in throughput of the selfish node gained via misbehaving is computed and, compared to the scenario when all nodes are genuine. Let t_g denote the throughput of node x when all nodes are genuine. Now, introducing one of the misbehaviors (say α-misbehavior with α = 0.5) at node x, let t_m denote the throughput of node x, when all other nodes are still genuine. Then, the effectiveness e of α-misbehavior for α = 0.5 is characterized as,

$$e = \frac{t_m - t_g}{t_g} * 100 \tag{4.1}$$

Indeed, e measures the magnitude of the incentive that a selfish user has in order to misbehave.

4.2 OPNET Simulation Setup

A scenario of the 802.11 wireless LAN protocol with 10 nodes in a 100 meter by 100 meter area is created, where 9 nodes are sending traffic to one receiver, and the distance between receiver and each of the senders is 30 meters. Figure 4.1 depicts the simulation scenario in OPNET. Out of the 9 senders, one sender is configured to misbehave according to the level and type of misbehavior desired. The size of each data packet equals 512 bytes, and the slot time is 20 microseconds. The heavy traffic load scenario corresponds to an exponential packet arrival rate of 100 packets per second.

The data rate of the network equals 2 Mbps. Both the high-load and medium-load scenarios overload the network beyond its capacity, by generating a total traffic load of 3.7 Mbps and 2.8 Mbps respectively. The low-load scenario corresponds to 0.9 Mbps, which is quite less compared to the network capacity. The effectiveness of various types of misbehaviors, for three different

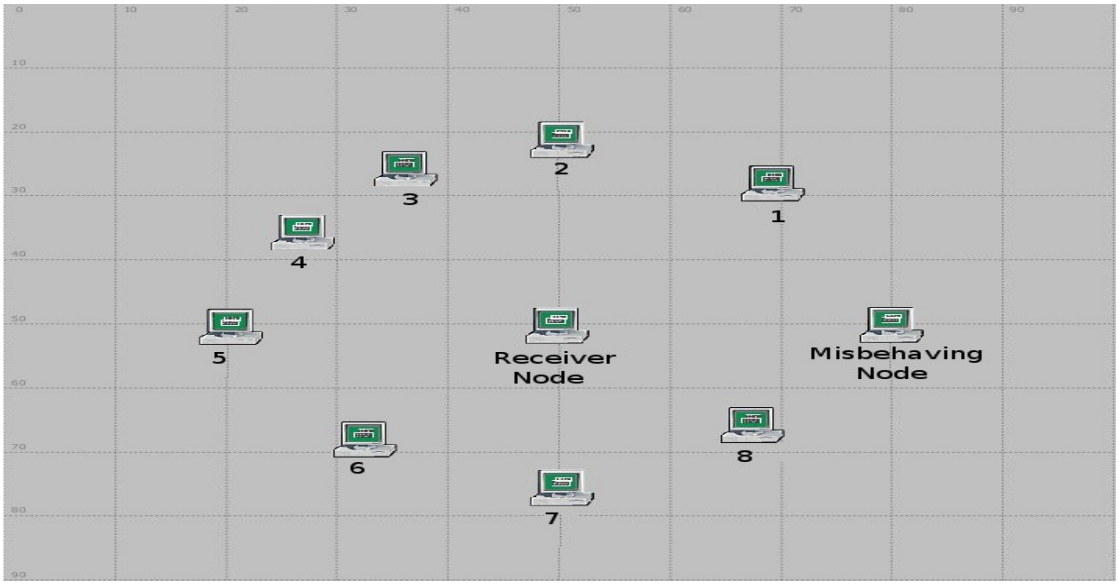

Figure 4.1 – OPNET scenario

traffic load scenarios, and at different levels of aggressiveness, are measured using simulation. Table 4.1 lists the 802.11 protocol parameters used for the simulations. Similarly, the medium load scenario corresponds to 77 packets per second, and the low-load scenario corresponds to 25 packets per second, as shown in Table 4.2.

Table 4.1 – Standard Parameters Used for the Simulation

Parameter	Value
MAC Protocol	802.11 b
Channel Bandwidth	2Mbps
Slot Time	20 Micro seconds
MAC Fragmentation Threshold	2304 bytes
Physical Characteristics	Direct Sequence
RTS Threshold	128 bytes

4.3 Simulation Results and Observations

Figure 4.2 depicts the effectiveness achieved using α- misbehavior at various values of α. It is observed that in the low-traffic load scenario, as the total LAN traffic is below the capacity, the throughput of each node remains the same at all values of α. Thus the selfish node does not gain

Table 4.2 – Simulation Scenario to Measure Effectiveness of Various Misbehaviors

Parameter	Value
Grid Size	100x100 meters
Packet Size	512 bytes
Low-Traffic Load	25 packets per second
Medium-Traffic Load	77 packets per second
High-Traffic Load	100 packets per second

much by misbehaving in this scenario. It is also observed that under the medium and high loads, there is a non-linear increase in misbehavior effectiveness with a decrease in the value of α below 1. However, this gain saturates after awhile (which corresponds to a selfish node being able to get all of its data across successfully), and further increasing the level of misbehavior does not lead to substantial throughput gains for the selfish node. The subsequent increase in misbehavior aggressiveness only degrades LAN throughput, which may raise an alarm for potential misbehavior detection. Therefore, a selfish node may choose to operate close to α = 0.4 for medium traffic, and near α = 0.3 for high traffic, in order to reap substantial throughput gains while hoping to avoid detection. Note that the selfish node could easily estimate the best value of α by noticing the diminishing returns as the level of misbehavior is increased further. All these observations hold true for other types misbehavior as well. Figure 4.3 depicts the effectiveness achieved using db-misbehavior at various values of deterministic backoff (*b*) chosen by the selfish node. It is observed that the results are quite similar to the case of α-misbehavior. Also it was noticed that under saturated traffic scenarios, and in the absence of misbehavior, the mean value of the backoff interval chosen by a genuine node over time $\approx$ 22. Therefore, there is a decrease in throughput for the selfish user for values of b significantly greater than 22. From the above two figures, it can be seen that the maximum throughput gain or effectiveness is limited to around 150$\%$ under medium traffic and around 230$\%$ under high traffic. Figure 4.4 depicts that for β-misbehavior, the increase in effectiveness is close to linear with a decrease in β for $1 < \beta < 2$. As β is decreased below 1, the misbehavior effectiveness increases non-linearly initially, particularly because the value of CW_{min} is also decreased. The effectiveness saturates however, as is the case with other misbehavior types.

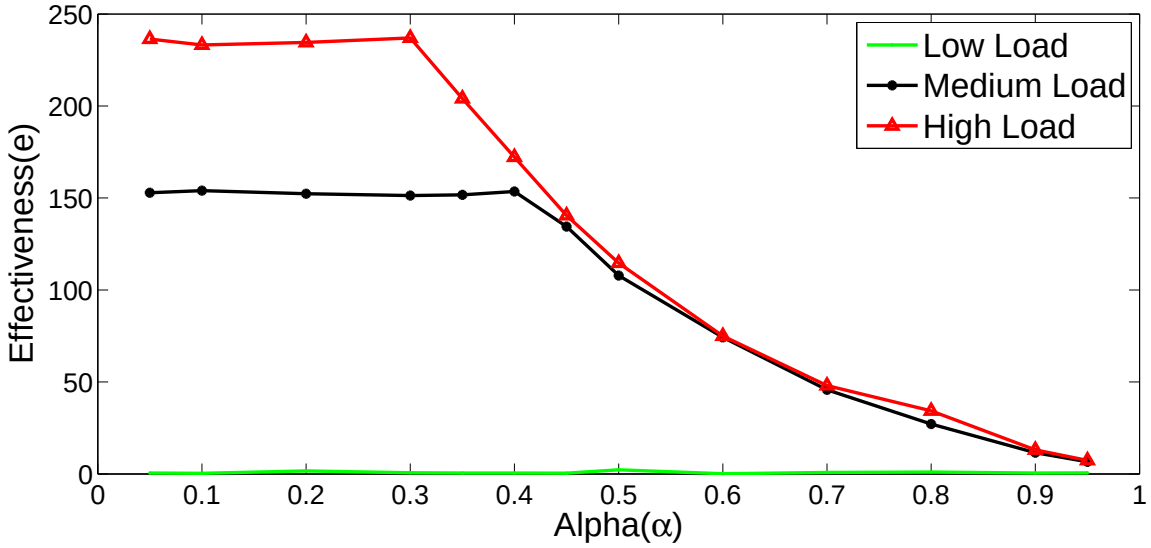

Figure 4.2 – Alpha(α) misbehavior

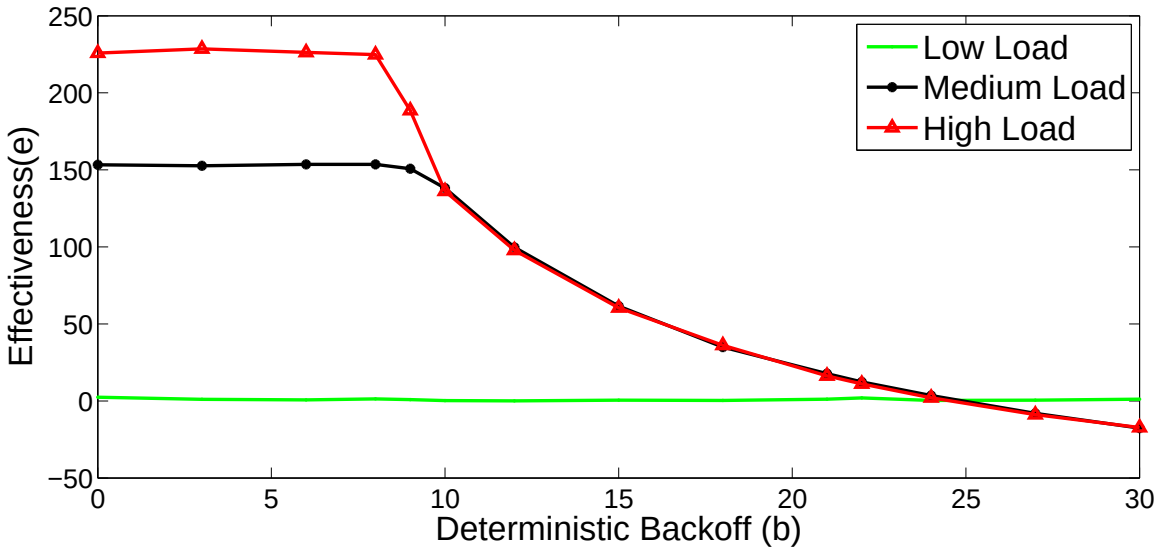

Figure 4.3 – Deterministic backoff (b) misbehavior

Figures 4.5 and 4.6 depict the effectiveness with CW_{max} and CW_{fix} misbehaviors, respectively. Under saturated traffic conditions with no misbehavior, the average value of contention window used by a genuine node ≈ 50. This leads to negative effectiveness for the selfish node when $\text{CW}_{fix} > 50$.

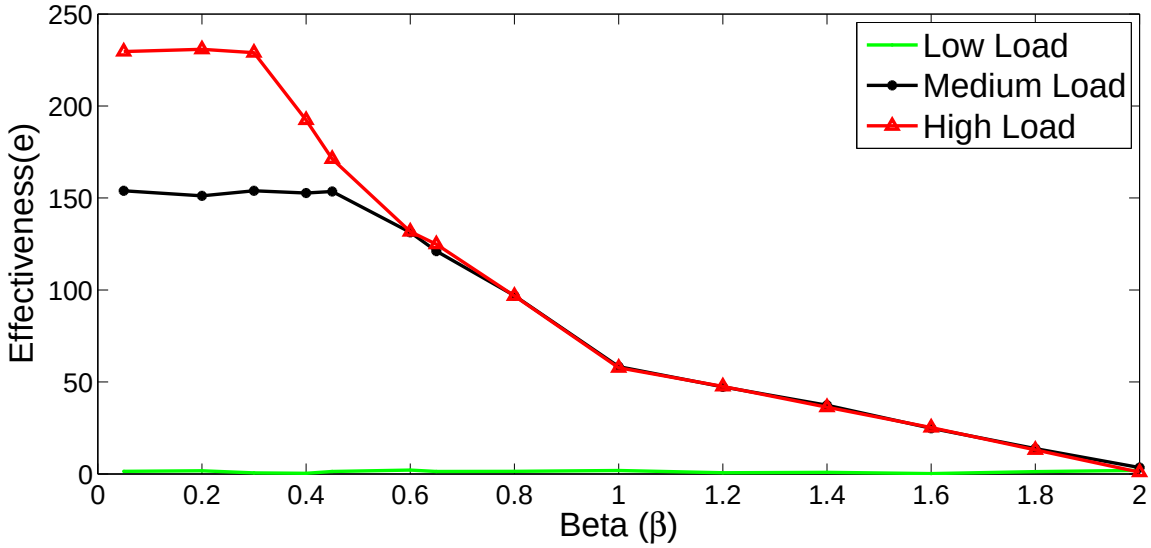

Figure 4.4 – Beta (β) misbehavior

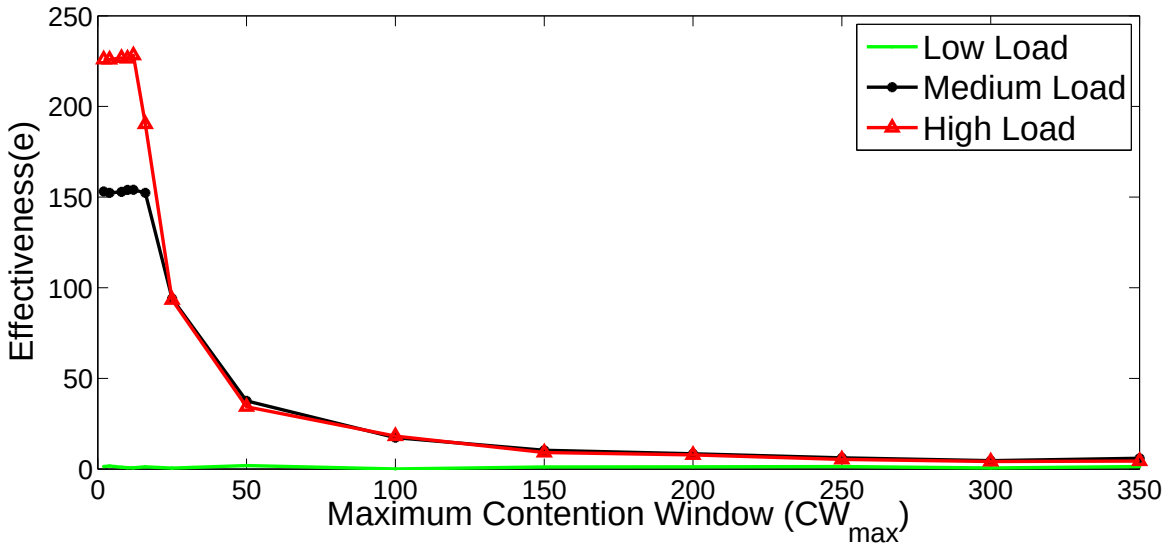

Figure 4.5 – CW$_{max}$ misbehavior

4.3.1 Hybrid Misbehaviors

Next, effectiveness achieved using a hybrid strategy is evaluated. Figures 4.7 and 4.8 correspond to α-misbehavior along with CW$_{max}$ and β, respectively. Note that saturation is reached at higher values of α compared with the α-misbehavior, however the maximum achievable effectiveness remains unchanged. Thus, a selfish node does not have any additional advantage to applying

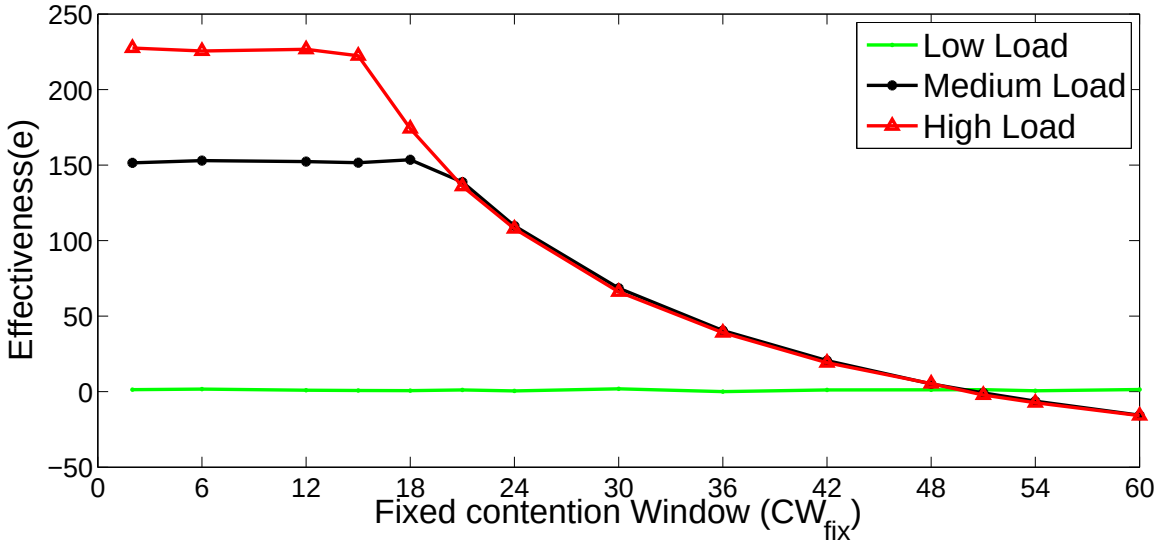

Figure 4.6 – CW_{fix} misbehavior

a hybrid strategy.

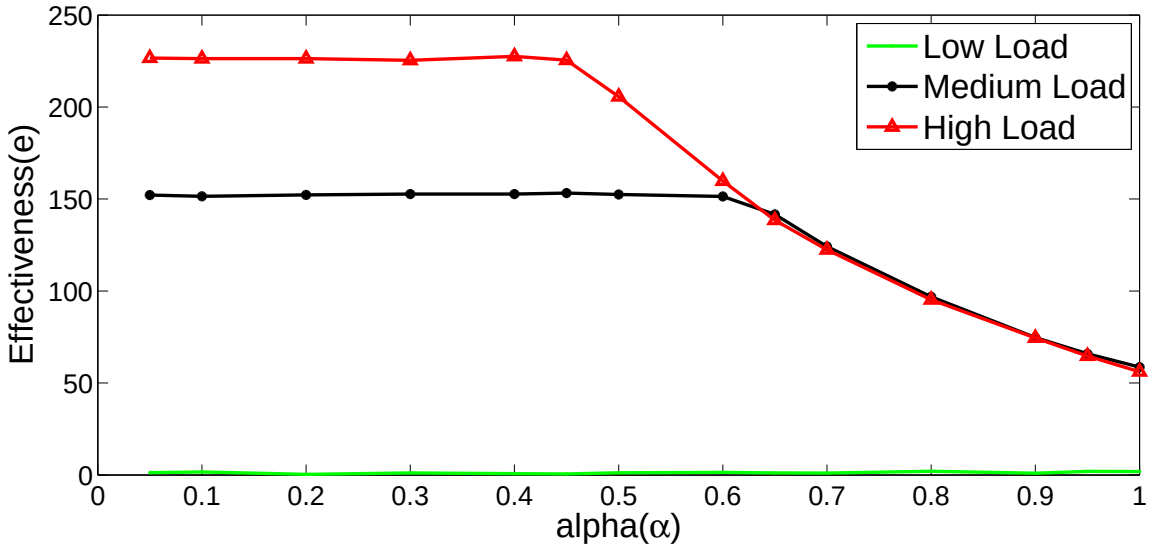

Figure 4.7 – Hybrid misbehavior with CW_{max} = 64

4.4 Analysis

In all of the misbehavior types considered, mild misbehaviors or low-traffic load do not lead to large improvements in the throughput of the selfish node. As the level of misbehavior is increased,

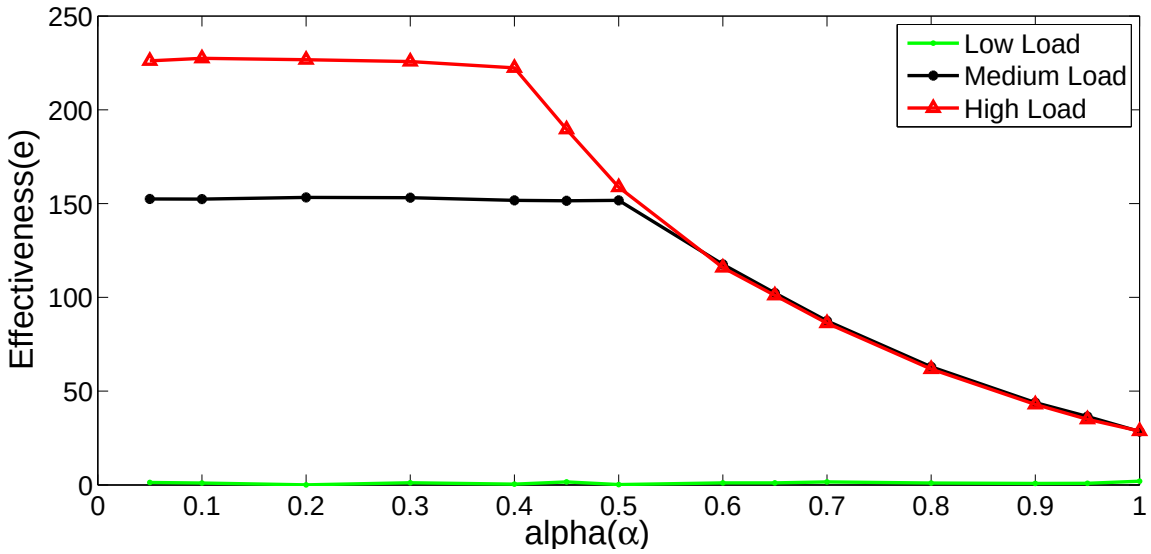

Figure 4.8 – Hybrid misbehavior with β = 1.5

from mild to aggressive, there is generally a non-linear increase in the effectiveness. However the effectiveness saturates around the region where the node misbehavior is quite aggressive. Thus a detection and reaction strategy should be employed only when the traffic load is considerably high and the level of misbehavior is reasonably aggressive. Also a selfish node could track its diminishing returns when increasing the aggressiveness of misbehavior, to behave in the region to reap maximum benefits while hoping to avoid detection.

4.5 Summary

This chapter summarizes the simulation of five types of misbehaviors using OPNET simulation software. Effectiveness is introduced to measure the incentive gained by the misbehaving node via misbehavior. Also plots that depict the effectiveness versus aggressiveness for three types of traffic loads are presented for each type of misbehavior. This chapter also discusses applying a hybrid misbehavior and observes the effectiveness of such misbehavior. Similarities between various types of misbehaviors at different traffic-load scenarios are identified. Scenarios where it is more appropriate to deploy a reaction strategy are identified.

CHAPTER 5

REACTION APPROACH

5.1 Effect of Misbehaviors on Fairness

Fairness in throughput achieved in the network is analyzed in the presence of misbehavior. In the high-traffic load scenario with nine senders, in the absence of any misbehavior, the average throughput of the eight genuine nodes is $\approx$ 145 Kbps.[3] Table 5.1 gives the throughput of eight genuine nodes for the scenario, where eight nodes are genuine senders and one misbehaving sender is employing α-misbehavior with α = 0.05. The throughput of the selfish node is observed to increase by 225$\%$ in such a scenario. The average throughput of the genuine nodes is $\approx$ 106.6Kbps, corresponding to a decrease of 26.5$\%$ each. Note that a cumulative decrease in throughput of the genuine nodes = 26.5 $\times$ 8 = 212$\%$ is slightly less than the increase in throughput for the selfish node. Thus, the presence of misbehavior causes overall LAN throughput to increase slightly. Also it is observed that the fairness in throughput of the genuine nodes is not affected due to the presence of the misbehavior. Using Jain's fairness index [14],[4] the fairness in the throughput of the nine genuine nodes in the absence of misbehavior is computed to be 0.9999 (a value of 1 corresponds to the maximum possible fairness index). In the presence of misbehavior, the fairness decreases to 0.622 due to the drastic increase in the throughput share of the selfish node. However, fairness in the throughputs of the eight genuine nodes in the presence of misbehavior equals 0.9998. This suggests that the structure of the CSMA/CA protocol, along with the RTS/CTS mechanism and randomly chosen backoff, is able to guarantee fairness among the genuine nodes even in the presence of a misbehaving node. *This is the intuition behind our proposed reaction strategy, wherein the genuine nodes, upon detecting the presence of misbehavior in the network, respond by collectively applying the same level of misbehavior* in order to guarantee a fair share of throughput for

[3]Note that OPNET throughput computations include a header of size 78 bytes for each packet, regardless of the packet size.

[4]Jain's Fairness Index: In a network with n nodes contending for channel access, Jain's fairness index(f) can be defined as
f = $\frac{[\sum_{i=1}^{n} x_i]^2}{n\sum_{i=1}^{n} x_i^2}$, where x_i is the throughput of node i.

Table 5.1 – Throughput of Genuine Nodes in Case of α-Misbehavior with α = 0.05

Node	Throughput in Kbps
1	105.343
2	107.194
3	106.608
4	107.596
5	108.542
6	104.298
7	105.793
8	107.774

all nodes in the network.

The comparison of effectiveness shown in Chapter 4 yields that most misbehaviors are quite effective in increasing the throughput share of selfish nodes. Next, the α and CW_{fix} misbehaviors are considered to study the effect of collective aggressive reaction strategy on throughput and fairness achieved in the network.

5.2 Collective Aggressive Reaction Strategy

The primary goal of the collective aggressive reaction strategy is to provide sufficient disincentive for the selfish node so that it does not try employing any misbehavior strategy. This could be achieved by triggering a reaction response by all genuine node's such that the selfish nodes throughput becomes less than what it would have been in the absence of any misbehavior. One approach to achieving this goal would be for the genuine nodes to accurately estimate the level of misbehavior of the selfish node, and try to replicate that misbehavior as a reaction response. It is shown that such a reaction response not only causes the selfish node's throughput effectiveness to decrease below 0 (thus providing the necessary disincentive), but also ensures fairness in the throughput of all nodes in the network. This result is in line with the earlier findings in section 5.1 which suggest that multiple nodes choosing backoff in a similar manner achieve almost equal share of the throughput. Assuming that the genuine nodes are able to detect the level of misbehavior in the network, the impact of the proposed reaction response on throughput and fairness

achieved in the network is analyzed. For the high load scenario with nine senders, Figure 5.1 depicts the overall LAN throughput achieved when all the nodes collectively apply α-misbehavior at various values of α. It is observed that employing the reaction response could degrade the overall

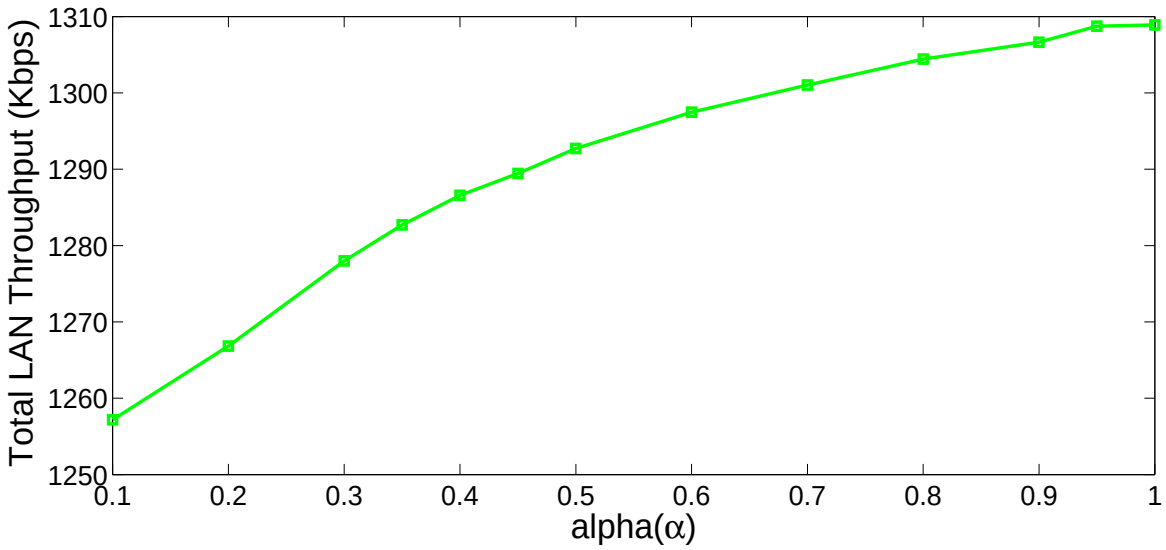

Figure 5.1 – Overall LAN throughput with collective α-misbehavior reaction response misbehavior

throughput, particularly at higher values of α. However, the selfish node's throughput also reduces to the levels available to other genuine nodes. Figure 5.2 depicts that the fairness index is close to optimal at all values of α. Figure 5.3 depicts the total LAN throughput when all the N nodes in the network collectively apply CW_{fix}-misbehavior for various values of N and CW_{fix}. Traffic parameters in this scenario are given in Table 5.2. When N = 5, this corresponds to a low-traffic load scenario. However, when N = 20, it corresponds to the high-load scenario. N = 15 is close to the medium-load scenario.

Table 5.2 – Traffic Scenario - N Nodes in the Network Collectively Apply CW_{fix}-Misbehavior

Parameter	Value
Packet Size	512 bytes
Packet Arrival Rate	45 packets per second

In the case of low traffic (N = 5), the degradation in overall throughput with increase in level of

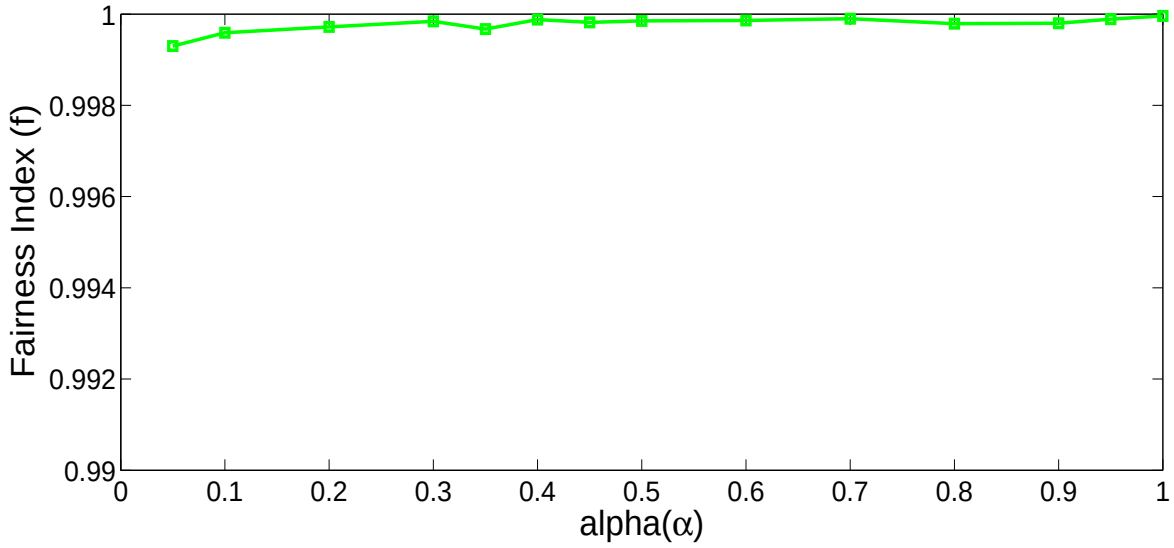

Figure 5.2 – Fairness index with collective (α)-misbehavior reaction response misbehavior

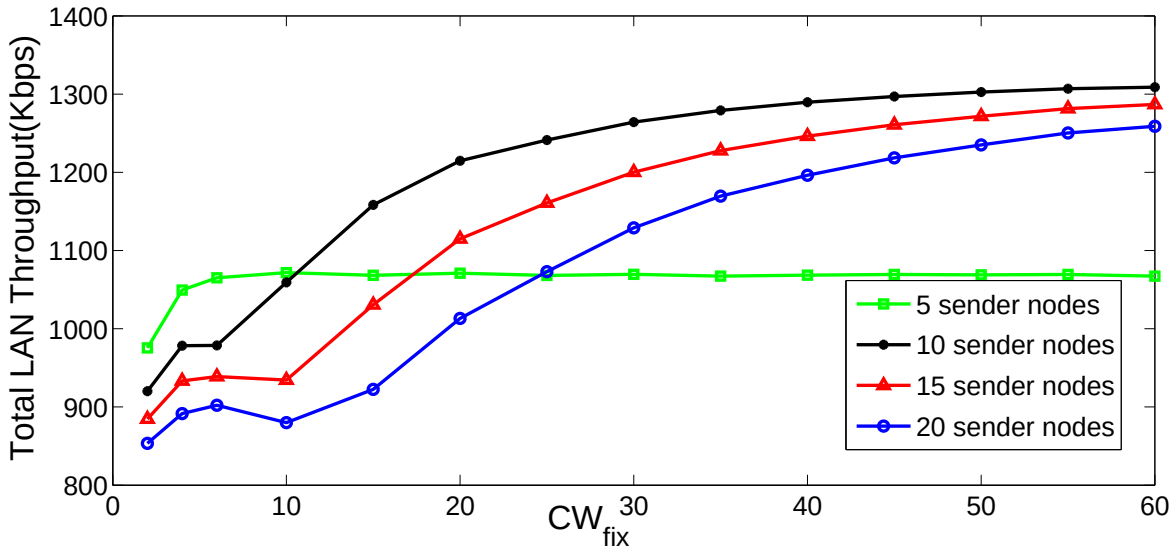

Figure 5.3 – Overall LAN throughput with collective CW$_{fix}$-misbehavior reaction response misbehavior

misbehavior is quite less, as expected. However, the impact of level of misbehavior on throughput degradation is higher as traffic load (or N) increases. For a particular value of CW$_{fix}$, and under saturated traffic conditions, the overall throughput decreases with an increase in load.

This difference is more prominent for reasonably aggressive choices of CW$_{fix}$ (12-25). In all scenarios, the nodes in the network are able to achieve a fair share of the throughput. Figure 5.4

depicts the share of throughput for the high-load scenario corresponding to the most aggressive reaction response, CW_{fix} = 2. The fairness index in this case equals 0.9997.

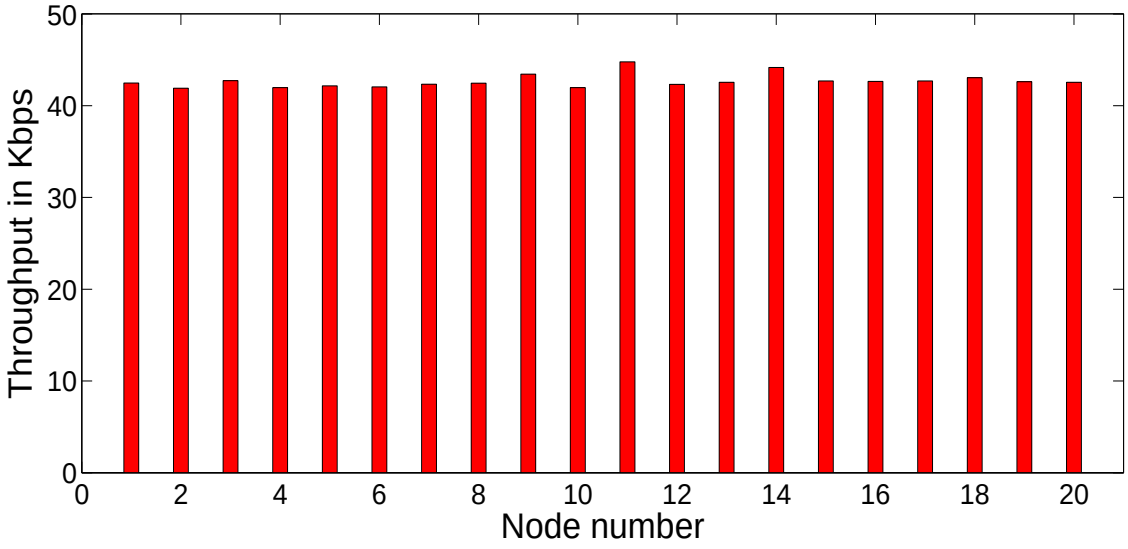

Figure 5.4 – Throughput share of all nodes with CW_{fix} = 2

5.3 Distributed Implementation of Reaction Approach

5.3.1 Estimation of Level of Misbehavior

In comparison with known detection and reaction schemes, the proposed aggressive reaction response in section 5.2 is similar to the meaningful Nash equilibrium outlined by Cagalj et al.[11] . However, if the level of misbehavior is the most aggressive, the reaction response converges towards the equilibrium corresponding to network collapse. The reaction scheme could be based upon the backoff values chosen [6] (or the throughputs observed [4]) of all nodes in the network over time. This section presents a reaction scheme based on the node's own throughput. However, the node does not need to know the throughput of the other nodes in the network.

5.3.2 Reaction Algorithm

The algorithm can be outlined as follows:

- Every node computes its own throughput T_x.
- It compares its throughput with that of the genuine throughput T_g.
- If a node finds its throughput less than a Threshold [5] then it triggers the collective aggressive reaction scheme.

The assumption that is made during the simulations is that a node always knows the genuine throughput T_g, which could be computed using Bianchi's analysis [15]. There are two goals that the algorithm is expected to achieve

- Primary goal is that throughput of the misbehaving node is less than that of T_g.
- Secondary goal is to achieve post-reaction fairness among the genuine nodes.

5.3.3 Network Software Tools

OPNET and NS-2 are two major software tools used to simulate networks. Although OPNET is a very powerful tool to simulate networks, NS-2 was also used in this thesis for following reasons: OPNET is not free ware and the frequent expiry of the licenses was troublesome, whereas NS-2 is free ware. OPNET documentation for the student versions is very limited and program APIs were not accessible to the user. On the other hand, NS-2 has very good documentation and all of the software can be modified as per the requirements of the user. Therefore, the reaction scheme is implemented in NS-2.

5.3.4 Comparison of OPNET and NS-2

This section provides the computation of genuine throughput for a channel with 1 Mbps and 2 Mbps capacity using Bianchi's [15] analysis and its comparison with that of NS-2's and OPNET's.[6] This is done to verify that T_g from the analysis closely matches that of the simulations. From

[5] Threshold is set to 80% of T_g in simulations.

[6] In both NS-2 and OPNET simulations, the traffic rate configured on the 2 Mbps channel is 3.68 Mb/s and on the 1 Mbps channel is 1.84 Mb/s, which is saturated traffic in the respective scenarios.

Bianchi's analysis [15], equations (5.2) for the transmission probability (τ) and equations (5.3) for the conditional collision probability (p) of a node are obtained. Solving the equations (5.2) and (5.3) using fixed point computation (C-program), values of p = 0.270320 and τ = 0.038627 are obtained. The parameters used are W = 32 (initial CW), m = 5 (number of backoff stages to reach CW_{max}) and n = 9 (number of sender nodes).

$$\tau = \frac{2(1-2p)}{(1-2p)(W+1) + pW(1-(2p)^m)} \tag{5.2}$$

$$p = 1 - (1-\tau)^{n-1}. \tag{5.3}$$

$$S = \frac{P_s P_{tr} E[P]}{(1-P_{tr})\sigma + P_s P_{tr} T_s + P_{tr}(1-P_s)T_c} \tag{5.4}$$

The values of P_{tr} and P_s are given by equations (5.5) and (5.6), respectively. P_{tr} is the probability that there will be at least one transmission in the given time slot and P_s is the probability the transmission occurring on the channel is successful, given there is at least one transmission. The values of P_{tr} and P_s computed for the scenario of nine senders are P_{tr} = 0.298 and P_s = 0.84.

$$P_{tr} = 1 - (1-\tau)^n. \tag{5.5}$$

$$P_s = \frac{n\tau(1-\tau)^{n-1}}{P_{tr}} \tag{5.6}$$

Table 5.3 – Overall Throughput of a 1 Mbps Channel with RTS/CTS Enabled

Number of Senders	Throughput Opnet (Mb)	Throughput NS-2 (Mb)	Bianchi's Normalized Throughput (Mb)
One Sender	0.67	0.694	0.65
Two Sender	0.70	0.70	0.69
Nine Sender	0.71	0.704	0.71

Table 5.4 – Overall Throughput of a 1 Mbps Channel with RTS/CTS not Enabled

Number of Senders	Throughput Opnet (Mb)	Throughput NS-2 (Mb)	Bianchi's Normalized Throughput (Mb)
One Sender	0.75	0.78	0.722
Two Sender	0.78	0.77	0.752
Nine Sender	0.713	0.695	0.699

Table 5.5 – Overall Throughput of a 2 Mbps Channel with RTS/CTS Enabled

Number of Senders	Throughput Opnet (Mb)	Throughput NS-2 (Mb)	Bianchi's Normalized Throughput (Mb)
One Sender	1.04	1.102	1.3
Two Sender	1.11	1.133	1.38
Nine Sender	1.141	1.128	1.42
Eighteen Sender	1.1	1.10	1.43

Table 5.6 – Overall Throughput of a 2 Mbps Channel with RTS/CTS not Enabled

Number of Senders	Throughput Opnet (Mb)	Throughput NS-2 (Mb)	Bianchi's Normalized Throughput (Mb)
One Sender	1.257	1.34	1.44
Two Sender	1.34	1.36	1.504
Nine Sender	1.265	1.235	1.398
Eighteen Sender	1.073	1.13	1.2968

E[P], the packet payload is 512 bytes or 4096 bits or 81.92 slot time units.[7] Using equation (5.4) and parameter values from Bianchi's paper [15], computing the normalized throughput S from equation (5.4), S = 0.719 (71.9%) is obtained. With a single-sender in the network value of τ is 0.0606. Hence, for the single-sender scenario P_{tr} = 0.0606 and P_s = 1. The value of S for single-sender scenario is 0.655 (65.5%). Increasing the senders to two, values of τ, P_{tr}, and P_s are 0.057044, 0.1108 and 0.693 (69.3%) respectively. Tables 5.3 through 5.6 compare throughput's of Bianchi's, OPNET and NS-2 for varying number of nodes in the network. Table 5.3 gives the overall throughput of a 1 Mbps channel with RTS/CTS enabled Table 5.4 gives the overall

[7] One slot time unit corresponds to 20μ sec

throughput of a 1 Mbps channel with RTS/CTS not enabled Table 5.5 gives the overall throughput of a 2 Mbps channel with RTS/CTS enabled and Table 5.6 gives the overall throughput of a 2 Mbps channel with RTS/CTS not enabled. Results show that the values from OPNET and NS-2 closely agree with each other, and with those obtained using analysis.

5.4 Simulation

The detection scheme provided in section 5.3.2 is simulated in NS-2. The parameters used for the simulations are the same as those used in OPNET simulations. Figure 5.5 depicts the layout of nodes in NS-2.

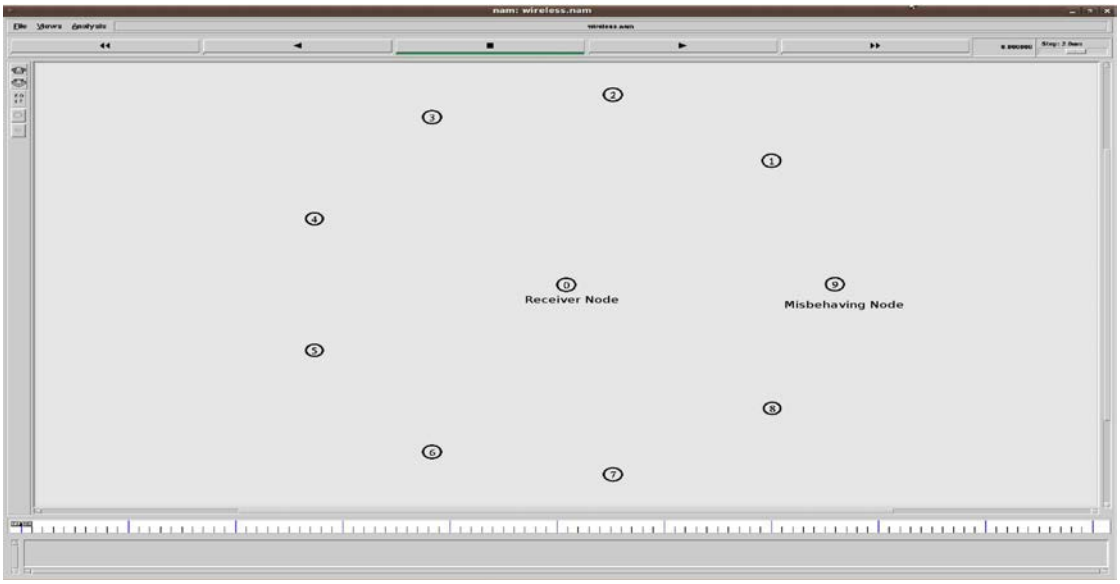

Figure 5.5 – Node scenario in NS-2

A single node with alpha (α) misbehavior with α = 0.1 (aggressive misbehavior) is used. To better understand the effect of the reaction response, high traffic corresponding to 100 packets per second is employed. The total traffic corresponding to all nine sender nodes is 3.68 Mb/s, which is higher than the channel capacity. The genuine throughput is computed by simulating the network with all genuine nodes. T_g is computed taking the average throughput of all genuine nodes and is $\approx$ 125 Kbps. Now the nodes compute their throughput T_x every second and compare it with that of Threshold. If throughput T_x is less than threshold, a node may employ any known misbehavior

as a reaction. The CW_{fix} misbehavior is used as the reaction scheme in the simulations.

5.4.1 Non-Adaptive Long-Term Reaction with Reaction Packet Threshold

A genuine node will check if its throughput is less than Threshold

- If yes, it will employ reaction.[8]
- The duration of the reaction is the time taken by the node to successfully transmit the Reaction Packet Threshold (Rea Pkt Threshold) number of packets.
- After the node has successfully transmitted the Rea Pkt Threshold number of packets, it will check again, if the throughput in the duration was less than Threshold, if yes then it will repeat the reaction.

If throughput is greater than Threshold

- It will employ BEB. (NO REACTION)
- BEB is employed only for the duration to send BEB Packet Threshold number of packets successfully.

A larger reaction time helps achieve goals outlined in section 5.3.2. This can be done by sufficiently choosing Rea Pkt Threshold larger than BEB Packet Threshold. Fixing the Threshold value to 100 Kbps and BEB Packet Threshold to 10 Packets Rea Pkt Threshold is varied to observe the effect of the reaction scheme.

Figure 5.6 gives the throughput of nine nodes in a scenario where all nodes are genuine. Figure 5.7 gives the pre-reaction throughput of the nodes, node 1 being the misbehaving node. These throughputs are taken at the end of *Pre-reaction simulation time*[9]. It is only after *Pre-reaction simulation time* of simulation that the genuine nodes apply the reaction strategy. Hence the pre reaction throughput will be the same for all the simulations as long as the misbehavior type and misbehavior parameter are constant. In the current simulations, α = 0.1 misbehavior is employed.

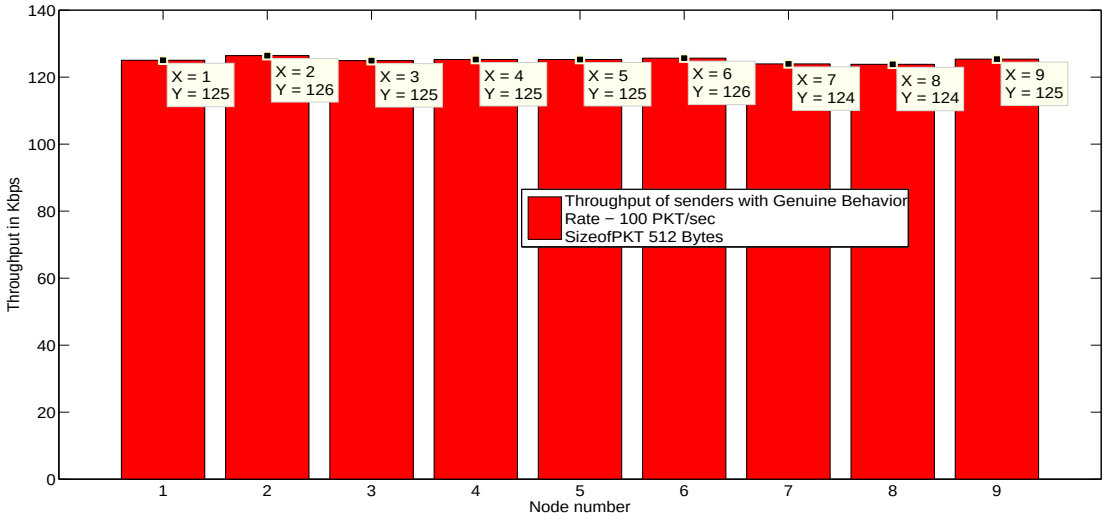

Figure 5.6 – Throughput of nodes with genuine behavior

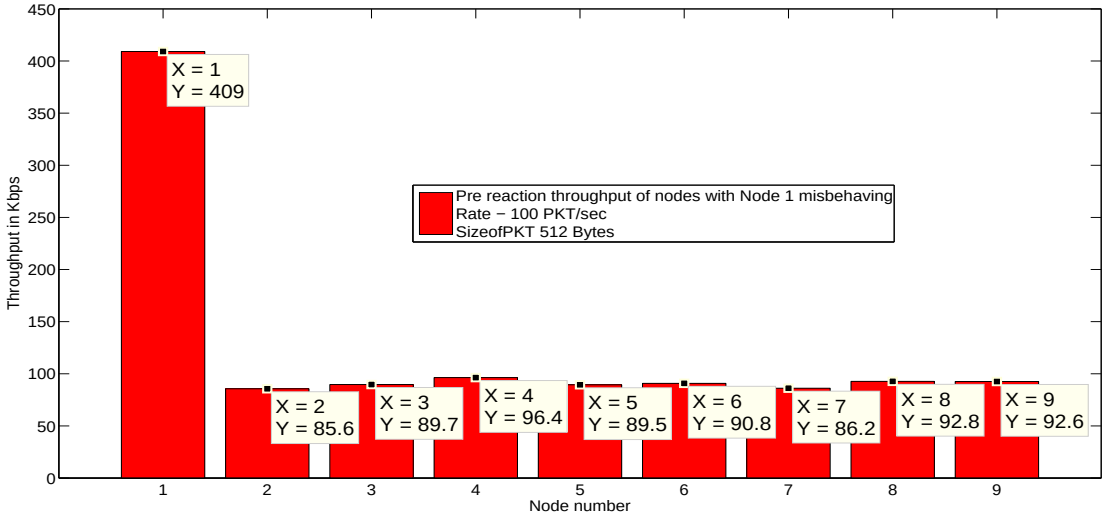

Figure 5.7 – Pre reaction throughput of nodes with alpha (α) = 0.1 misbehavior

Figure 5.8 gives the post-reaction[10] response when the Reaction Packet Threshold = 100 and CW_{fix} = 2. It can be seen that there is considerable decrease in the throughput of the misbehaving nodes. Figures 5.9 and 5.10 depict the post-reaction throughput for Rea Pkt Threshold of 200

[8]Reaction corresponds to usage of CW_{fix} misbehavior

[9]All the simulations consider a simulation time of 900sec. *Pre reaction simulation time* = 200 sec

[10]Post-reaction throughput corresponds to the throughput computed only for the interval in which reaction is applied. In this case it is from 200 sec to 900 sec.

and 250, respectively, with CW$_{fix}$ = 2 reaction. In both of these scenarios considerable decrease in the misbehaving node's throughput is observed. At Rea Pkt Threshold = 300 and CW$_{fix}$ = 2 reaction, fairness amongst the nodes is observed as shown in figure 5.11. Increasing Rea Pkt Threshold further to 500 at CW$_{fix}$ = 2 reaction, it is observed that misbehaving nodes throughput is less than that of genuine node's throughput as shown in figure 5.12. Choosing this value of Rea Pkt Threshold = 500, CW$_{fix}$ is varied as shown in figures 5.13 to 5.16. Increase in the CW$_{fix}$ corresponds to decrease in the aggressiveness of the reaction. At a value of CW$_{fix}$ = 10, near fair throughput amongst the nodes is achieved. For reaction using a given tuple, Table 5.7 gives the times spent by a genuine node employing aggressive backoff and BEB. Table 5.8 gives the percentage number of transmission attempts of a genuine node employing aggressive backoff and BEB. It can be observed that, with an increase in Rea Pkt Threshold, there is also increase in the percentage amount of time aggressive backoff is used. Looking at the throughputs of the nodes, it can be observed that with tuples having higher reaction times (percentage amount of time aggressive backoff is used), the goals of the reaction scheme are achieved subsection 5.3.2.

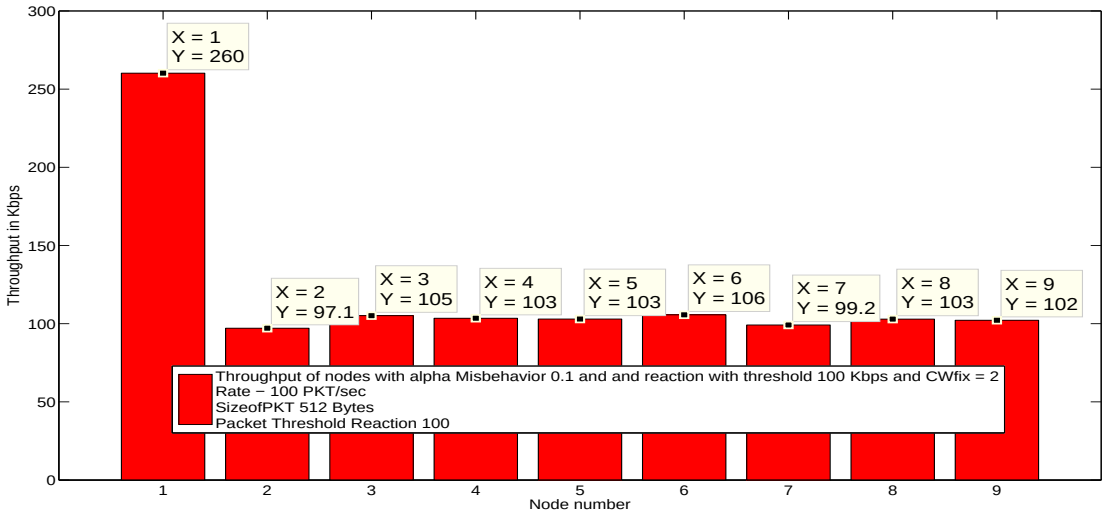

Figure 5.8 – Post reaction throughput of nodes with alpha(α) = 0.1 misbehavior and CW$_{fix}$ = 2 reaction and threshold 100 Kbps with reaction packet threshold 100

Table 5.7 – Fraction of Times Backoff is Chosen in Reaction with Rea Pkt Threshold

Reaction Packet Threshold, CW_{fix}	Fraction of Time Aggressive Backoff Used	Fraction of Time BEB Used
100,2	10.223572	89.776428
200,2	15.127650	84.872350
250,2	19.621560	80.378440
300,2	21.688844	78.311156
500,2	27.679277	72.320723
500,4	32.563546	67.436454
500,6	51.093825	48.906175
500,8	59.209549	40.790451
500,10	71.482800	28.517200

Table 5.8 – Fraction of Transmission Attempts with Rea Pkt Threshold

Reaction Packet Threshold, CW_{fix}	Fraction of Aggressive Transmission Attempts	Fraction of BEB Attempts
100,2	93.872413	6.127587
200,2	97.153495	2.846505
250,2	97.924499	2.075501
300,2	98.188364	1.811636
500,2	98.914258	1.085742
500,4	97.903544	2.096456
500,6	97.997417	2.002583
500,8	98.164800	1.835200
500,10	98.331041	1.668959

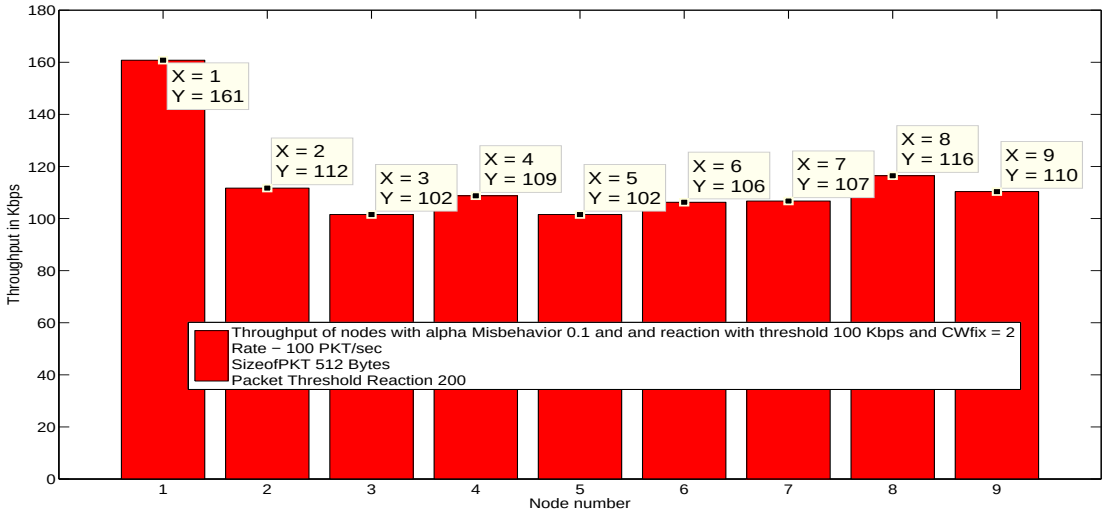

Figure 5.9 – Post-reaction throughput of nodes with alpha(α) = 0.1 misbehavior and CW$_{fix}$ = 2 reaction and threshold 100 Kbps with Rea Pkt Threshold 200

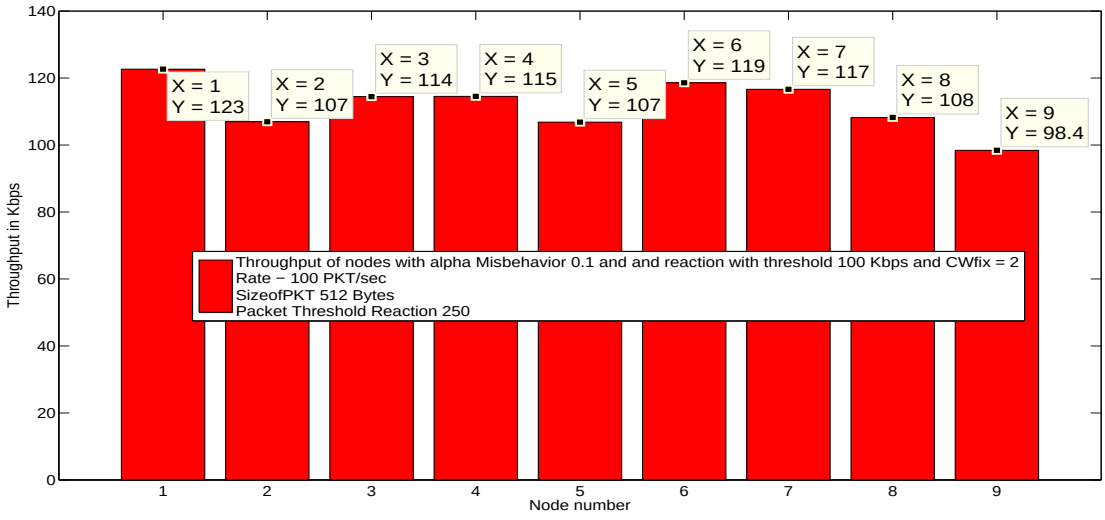

Figure 5.10 – Post-reaction throughput of nodes with alpha(α) = 0.1 misbehavior and CW$_{fix}$ = 2 reaction and threshold 100 Kbps with Rea Pkt Threshold 250

5.4.2 Adaptive Reaction With Rea Interval

The choice of Rea Pkt Threshold and CW$_{fix}$ were not dynamic in long-term reaction considered in section 5.4.1. This section provides a dynamic adaptive algorithm that will determine the values of Rea Pkt Threshold and CW$_{fix}$. The reaction is primarily dependent on these parameters -

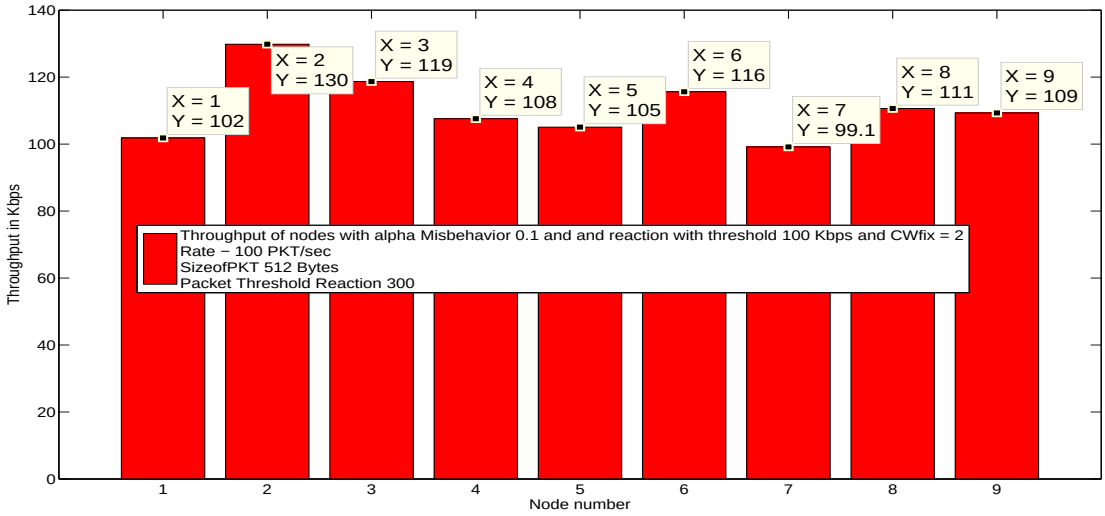

Figure 5.11 – Post-reaction throughput of nodes with alpha(α) = 0.1 misbehavior and CW_{fix} = 2 reaction and threshold 100 Kbps with Rea Pkt Threshold 300

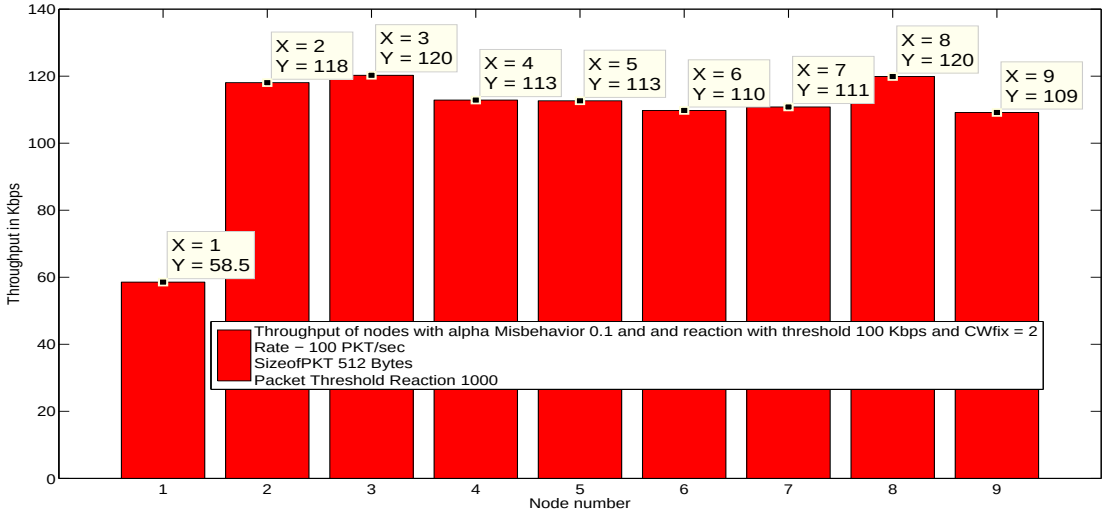

Figure 5.12 – Post-reaction throughput of nodes with alpha(α) = 0.1 misbehavior and CW_{fix} = 2 reaction and threshold 100 Kbps with Rea Pkt Threshold 500

CW_{fix} used for reaction, Rea Pkt Threshold, BEB Pkt Threshold and Threshold. Initially the genuine nodes do not apply any reaction policy. It is only after Pre Reaction Simulation Time amount of time they apply the reaction. The reaction being the usage of CW_{fix} misbehavior. This reaction is employed only if the throughput of the node is below the Threshold. Now when a node starts the

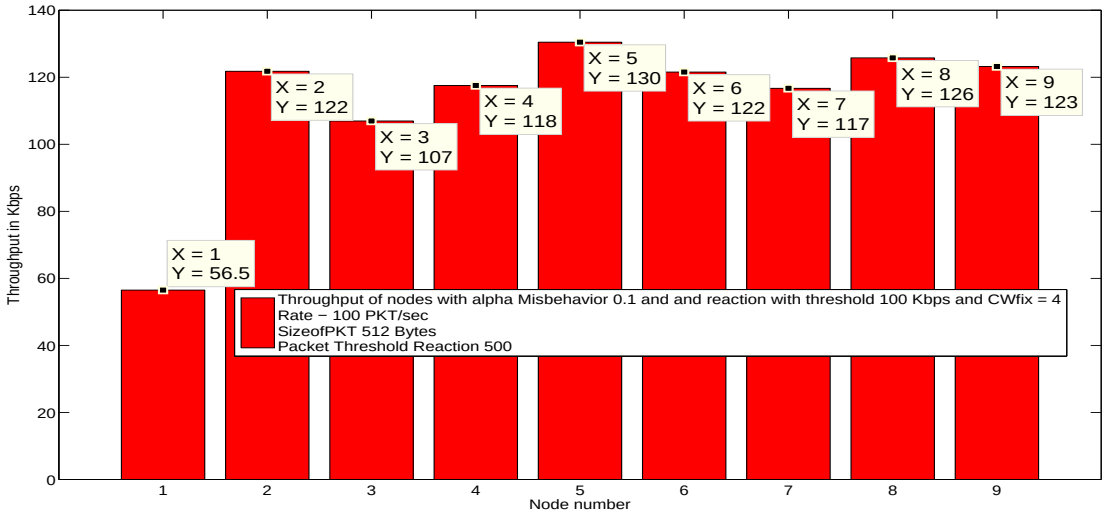

Figure 5.13 – Post-reaction throughput of nodes with alpha(α) = 0.1 misbehavior and CW_{fix} = 4 reaction and threshold 100 Kbps with Rea Pkt Threshold 500

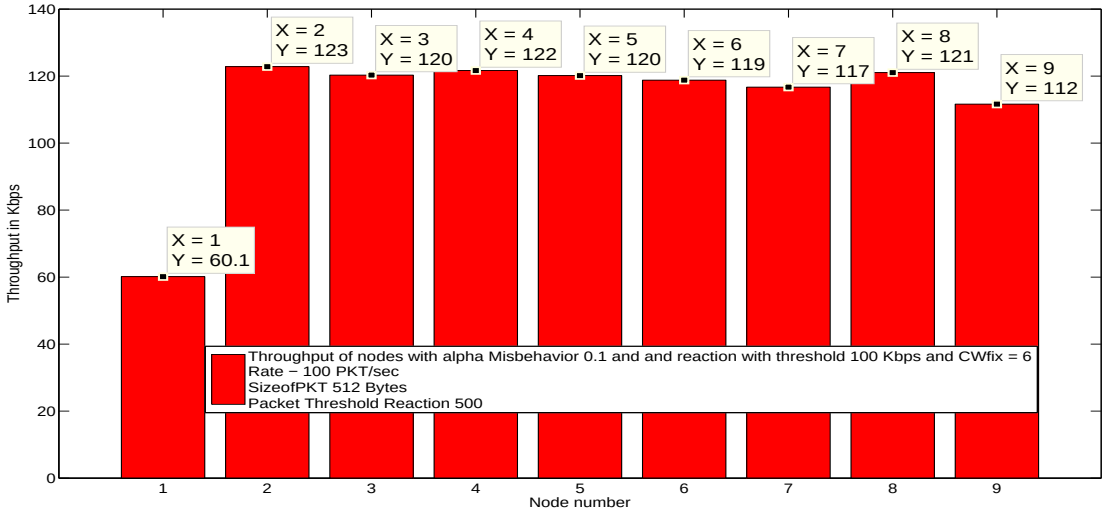

Figure 5.14 – Post-reaction throughput of nodes with alpha(α) = 0.1 misbehavior and CW_{fix} = 6 reaction and threshold 100 Kbps with Rea Pkt Threshold 500

reaction it will employ the reaction until it has Rea Pkt Threshold number of packets successfully transmitted. This period of time is represented as Y. After the Y period of time the node again checks for its throughput achieved over the Y period of time. If it is less than the Threshold, then it again employs the reaction; otherwise employs BEB. BEB is only employed for the time BEB

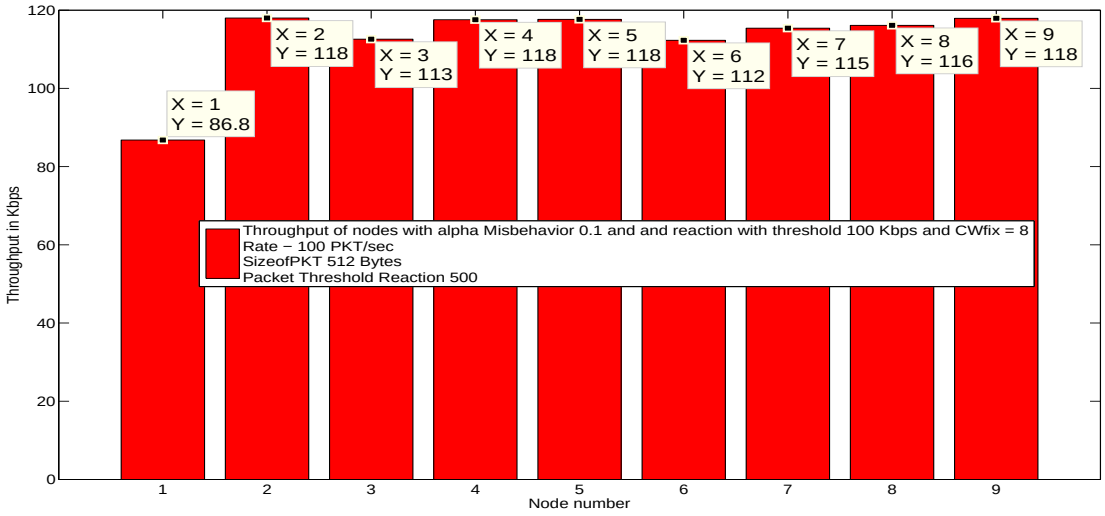

Figure 5.15 – Post-reaction throughput of nodes with alpha(α) = 0.1 misbehavior and CW_{fix} = 8 reaction and threshold 100 Kbps with Rea Pkt Threshold 500

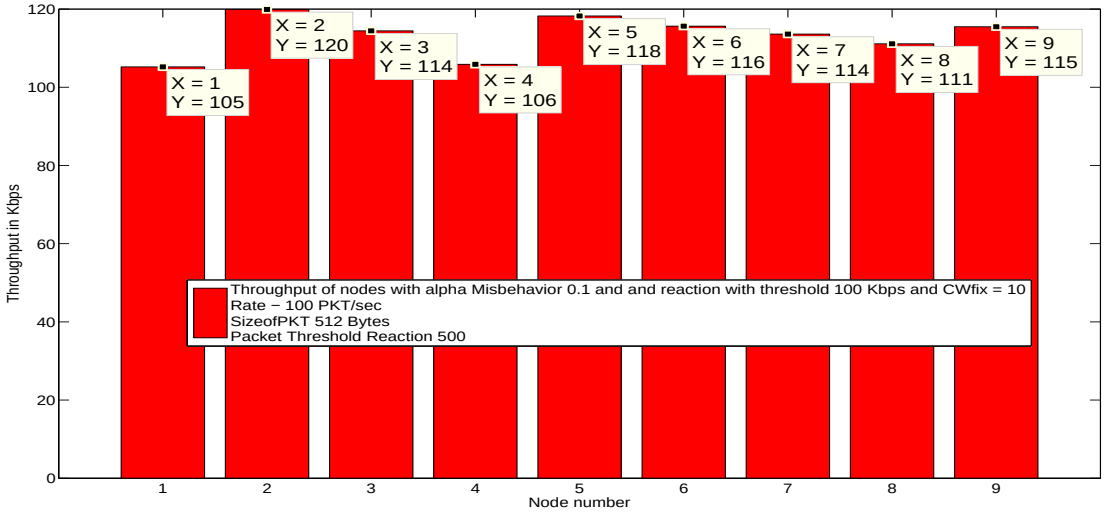

Figure 5.16 – Post-Reaction throughput of nodes with alpha(α) = 0.1 misbehavior and CW_{fix} = 10 reaction and threshold 100 Kbps with Rea Pkt Threshold 500

Pkt Threshold number of packets are successfully sent and this period is represented by N. Again the throughput achieved during the N period of time is computed and checked with Threshold and the process continues. Figure 5.17 depicts the working of the reaction scheme in a scenario where throughput over the reaction period of time (Y) is always greater than threshold. The adaptive

algorithm used for computation of Rea Pkt Threshold is as follows[11].

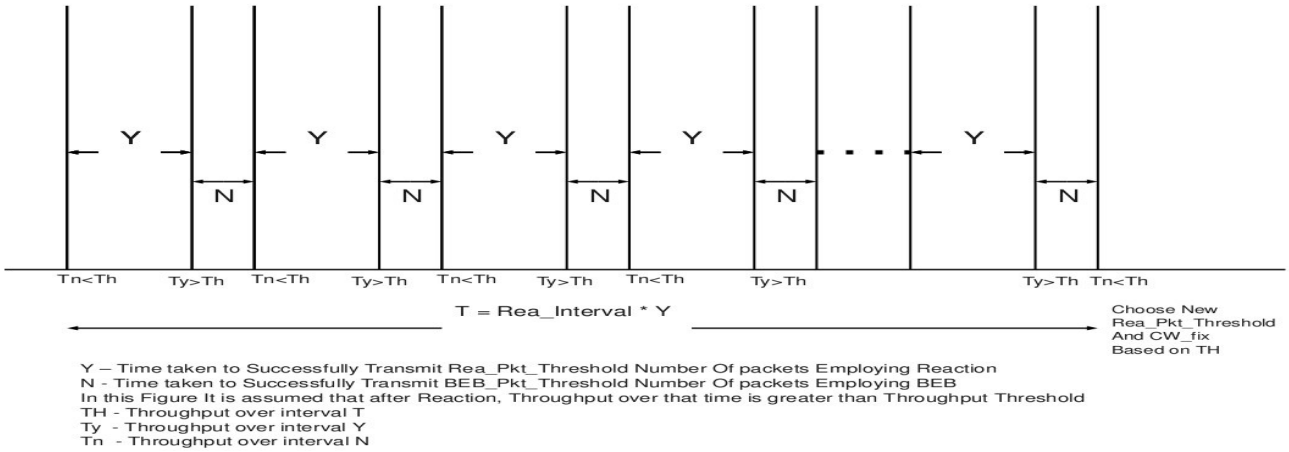

Figure 5.17 – How the Reaction Scheme Works

1. Start the reaction with tuple (Rea Pkt Threshold, CW_{fix}) = (100, 16).

2. Use this scheme for a period of time T. If a fixed value tuple of (Rea Pkt Threshold, CW_{fix}) Y is the time taken to send Rea Pkt Threshold number of packets, then T corresponds to Rea_interval number of Y periods. A Rea_interval value of 25 is used in the current algorithm.

3. Check the average throughput TH in the past T time period. If this average throughput is less than 125 Kbps (T_g), then increase the Rea Pkt Threshold and decrease the CW_{fix} as provided in the algorithm 5.4.3. Otherwise, do not change the tuple (Reaction Packet Threshold, CW_{fix}), and go to Step 2.

[11] BEB Pkt Threshold being a constant value of 10

5.4.3 Adaptive Algorithm

The adaptive algorithm for reaction with Rea-interval can be outlined as follows:

*If (TH $\leq$ T_g*3/4)*

*/*TH $<$ 75% of T_g */*

*Rea Pkt Thr = Rea Pkt Thr *2; CW_{fix} = CW_{fix}/2; if($CW_{fix} < 2$) CW_{fix}=2*

/ 2, 0.5 */*

*else if ((TH $>$ T_g*3/4) && (TH $\leq$ T_g*4/5))*

/ TH $>$ 75% and TH $\leq$ 80% of T_g */*

*Rea Pkt Thr = Rea Pkt Thr *3/2; CW_{fix} = CW_{fix}*2/3; if($CW_{fix} < 2$) CW_{fix}=2*

** 1.5, 0.67 **

*else if ((TH $>$ T_g*4/5) && (TH $\leq$ T_g*9/10))*

/ TH $>$ 80% and TH $\leq$ 90% of T_g */*

Rea Pkt Thr = Rea Pkt Thr + 10; CW_{fix} = CW_{fix} - 1; if($CW_{fix} < 2$) CW_{fix}=2

*else if (TH $>$ T_g + Delta) /*Delta = 10*/*

/ TH $>$ 100% of T_g */*

Rea Pkt Thr = Rea Pkt Thr - 20; CW_{fix} = CW_{fix} + 1;

/ else Tuple remains unchanged */*

Go to step 2.

5.4.4 Convergence

After every T period of time, each node runs the above algorithm. Depending on throughput achieved over the T period of time, the node chooses the new tuple (Rea Pkt Threshold, CW_{fix}). If the throughput value is between T_g*0.9 and (T_g + Delta), then the node will use the same

tuple. These instances when the node runs the algorithm are called *Decision Instances*. If a node uses the same tuple for three decision instances, then the node will converge to that tuple, and thereafter employ that tuple. If all nodes in the network converge to a tuple, then it is said that the convergence has been achieved, and post-convergence throughput is then measured.[12] Figure 5.18 shows the post-reaction throughput of nodes in the network, and figure 5.19 shows the post-convergence throughput of the network. Figures 5.20 and 5.21 depict the Rea Pkt Threshold and CW_{fix} values, respectively, chosen by a node, while executing the adaptive algorithm respectively. It can be observed that the node converges to the tuple, when the tuple remains unchanged for three decision instances. The goals of the reaction scheme, as mentioned in section 5.3.2, are achieved when the following occurs:

- The Throughput of the misbehaving node is less than that of T_g (125 Kbps).
- Jain's fairness index of 0.9988 has been achieved among the genuine nodes for the post-convergence throughputs.

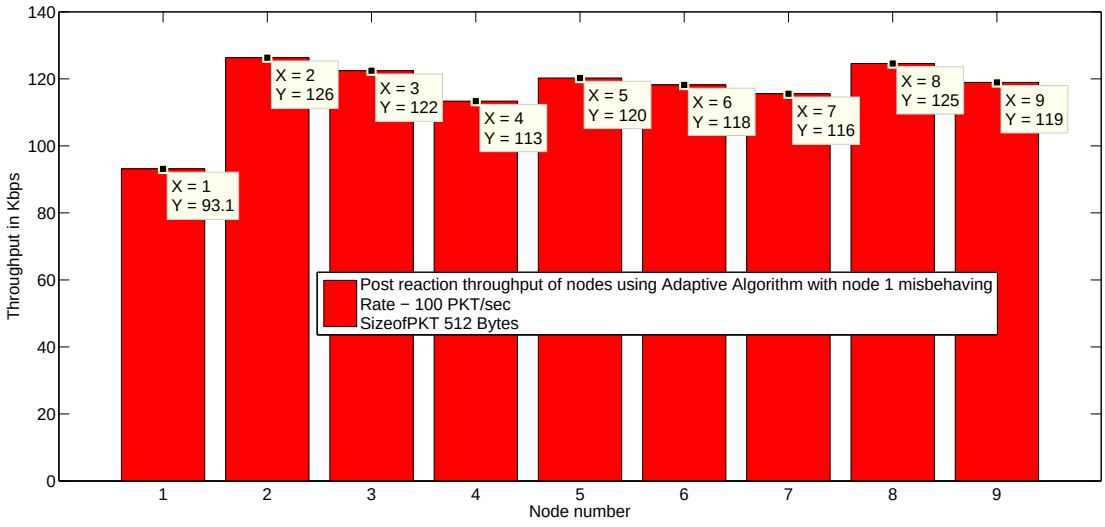

Figure 5.18 – Post-reaction throughput of nodes with adaptive algorithm

[12]Post-convergence throughput is throughput that is achieved by a node after convergence has occurred.

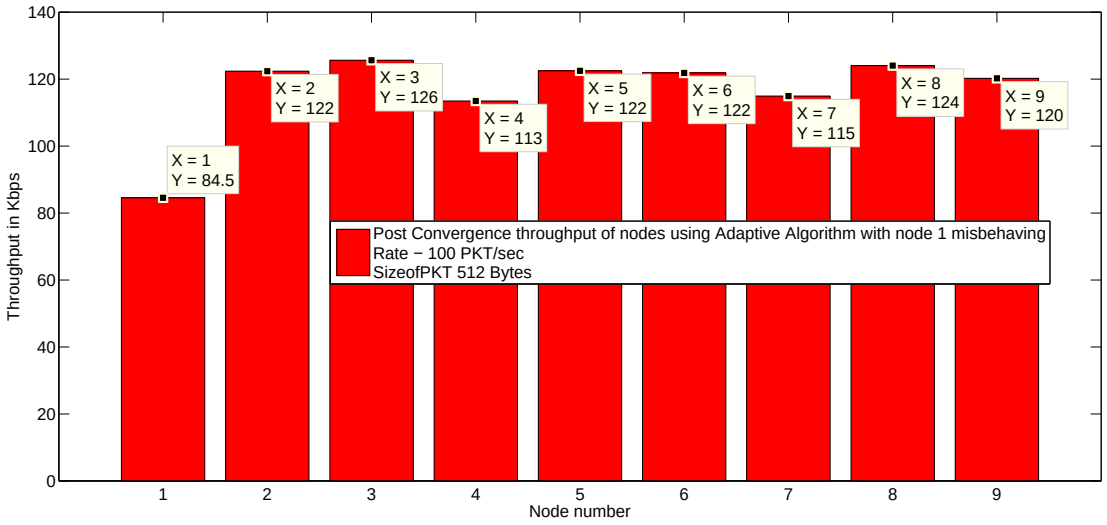

Figure 5.19 – Post-convergence throughput of nodes with adaptive algorithm

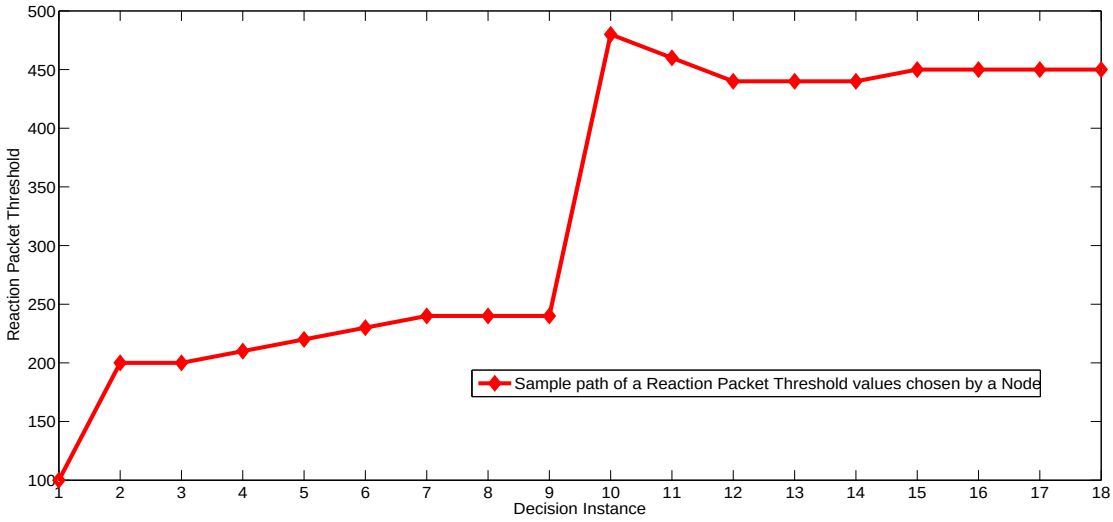

Figure 5.20 – Rea Pkt Threshold sample path towards convergence of a genuine node while reacting with adaptive algorithm

5.5 Summary

This chapter analyzes the effect of the collective aggressiveness of genuine nodes on the fairness achieved in the network. A novel collective aggressive reaction approach to offset the effect of misbehaviors in a network is then proposed. Simulation results and plots are provided to sup-

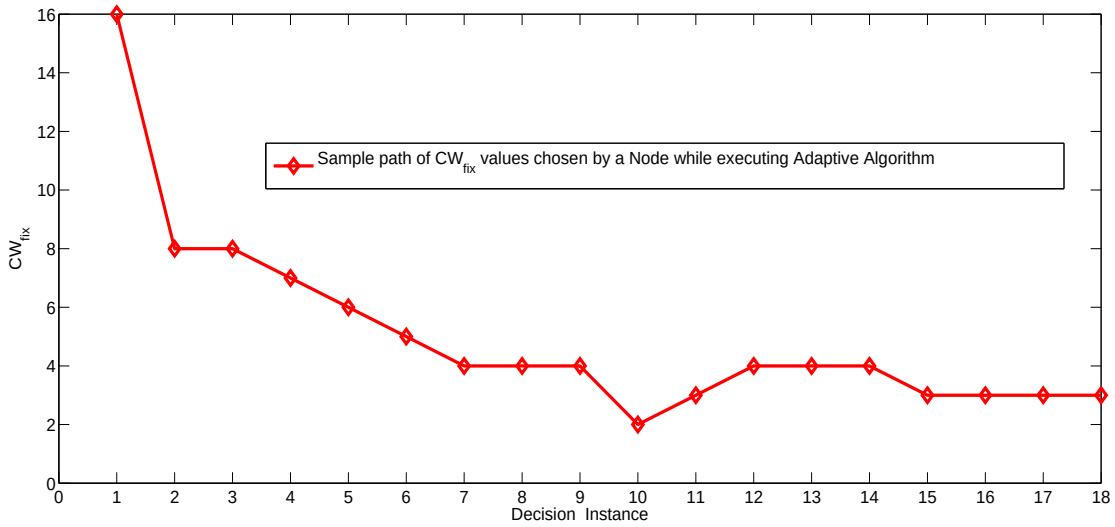

Figure 5.21 – CW_{fix} sample path towards convergence of a genuine node while reacting with adaptive algorithm

port the applicability of the reaction approach. An adaptive dynamic algorithm is implemented and results are presented and results that ensure the working of the algorithm are presented.

CHAPTER 6

CONCLUSION

This thesis classifies five types of MAC layer misbehaviors- α, β, CW_{fix}, CW_{max}, and Deterministic backoff(db), that are achieved via modifications to the IEEE 802.11 BEB algorithm. Impact of these five misbehaviors on throughput and fairness of a network, under various traffic load scenarios is studied. An effectiveness measure is defined to identify the key characteristics of the misbehaviors. Effectiveness curves are then plotted and common characteristics among all the misbehavior types are pin pointed and analyzed. Need for a reaction scheme to mitigate the misbehaviors in a network is realized and two goals are defined that any reaction scheme designed is expected to achieve. The primary goal being the reduction of effectiveness of misbehavior either to zero or negative, and the second to ensure post-reaction fairness in the network.

Based on observed effect of misbehaviors on the network, a collective aggressive reaction policy is proposed, which ensures fairness in the network at the expense of overall network throughput degradation. Simulating the misbehaviors using OPNET simulation tool, it is shown that the reaction scheme indeed reduces the effectiveness of the misbehaviors in the network and also ensures post reaction fairness among the genuine nodes.

A non adaptive long term reaction with reaction packet threshold is then proposed to refine the collective aggression reaction. Using CW_{fix} misbehavior as reaction and varying the associated parameters- Rea Pkt Threshold and BEB Pkt Threshold of reaction scheme, it is shown using NS-2 simulations that the long term reaction scheme achieves the primary goals of the reaction scheme at sufficiently high values of Rea Pkt Threshold. Extending the long term reaction by introducing Rea Interval and an adaptive algorithm, a distributed implementation of the long term reaction is proposed. Based on the throughput of a node over every Rea Interval, the usage of tuple (Rea Pkt Threshold, CW_{fix}) is varied dynamically. Convergence criteria is then defined, which takes care of the fact that nodes should employ sufficiently aggressive tuples to achieve the goals of a reaction scheme. Simulating the adaptive reaction in NS-2,

- The effectiveness of the misbehaving node is brought down to negative.

- Jains fairness index of 0.9988 is achieved among the genuine nodes for the post convergence throughputs.

Future work could include the extension of the current algorithm which allows the genuine nodes to take care of the following:

- Misbehaving node leaving the network.
- Computation of genuine throughput T_g dynamically depending on the number of nodes in the network.

REFERENCES

REFERENCES

[1] S. Radosavac, J.S. Baras, and I.koutsopoulos "A Framework for MAC Protocol Misbehavior Detection in Wireless Networks," in *Proceedings of the 4th ACM Workshop on Wireless Security*, pp. 33-45, Cologne, 2000.

[2] 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*, pp. 255-265, Massachusetts, 2000.

[3] M. Raya, J-P. Hubaux, and I. Aad, "DOMINO: A system to Detect Greedy Behavior in IEEE 802.11 Hotspots," in *IEEE transacctions on mobile computing*, vol. 5, no. 12, pp. 1691-1705, December 2006.

[4] P. Kyasanur, and N. H. Vaidya, "Selfish MAC Layer Misbehavior in Wireless Networks," in *IEEE Transactions on Mobile Computing, vol. 4 , issue. 5*, pp. 502-516, September 2005.

[5] L. Guang, C. Assi, and Y. Ye, "DREAM: A System For the Detection and Reaction Against MAC Layer Misbehavior in Ad Hoc Networks," in *Computer Communications*, vol. 30, issue 8, pp. 1841-1853, June 2007.

[6] J-G. Lee, A. Tang, J. Huang, M. Chiang, and A. R. Calderbank, "Reverse-Engineering MAC: A Non- Cooperative Game Model," in *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 6, August 2007.

[7] "Wireless LAN control (MAC) and Physical Layer (PHY) Specification," *IEEE Std. 802.11, 1997*.

[8] Y. Rong, S.K. Lee, and H.A. Choi, "Detecting Stations Cheating on Backoff Rules in 802.11 Networks Using Sequential Analysis," in *Proc. 25th IEEE International Conference on Computer Communications (INFOCOM)*, Barcelona, Spain, April 2006.

[9] A. A. Cardenas, S. Radosavac, and J. S. Baras "Detection and Prevention of MAC Layer Misbehavior in Ad Hoc Networks," in *Proceedings of the 2nd ACM workshop on Security of Ad hoc and Sensor networks*, pp. 1-8, Washington, October 2004.

[10] L. Guang, C. Assi, and A. Benslimane, "Modelling and Analysis of Predictable Random Backoff in Selfish Environments" in *Proc. 9 ACM international symposium on Modeling analysis and simulation of wireless and mobile systems,*,Terromolinos, Spain 2006, pp. 86-90.

[11] M. Cagalj, S. Ganeriwal, I. Aad, and J.P. Hubaux, "" On Selfish Behavior in CSMA/CA Networks" in *Proc. 24th IEEE International Conference on Computer Communications(INFOCOM)*, Miami, USA, March 2005, pp. 2513 - 2524.

[12] P. Brenner, "A Technical Tutorial on IEEE 802.11 Protocol," in *BreezeCOM*, July 1996.

[13] V.R. Giri, and N. Jaggi, "MAC Layer Misbehavior Effectiveness and Collective Aggressive Reaction Approach" in *33rd IEEE Sarnoff Symposium, 2010*, pp. 1 - 5, Princeton, April 2010.

[14] R. Jain, W. Hawe, and D. Chiu, "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems," DEC-TR-301, September 26, 1984.

[15] G. Bianchi, "Performance Analysis of IEEE 802.11 Distributed Coordination Function" in *IEEE J. Sel. Areas Commun*, vol. 18, no. 3, pp. 535-547, Mar. 2000.

APPENDIX

CODE USED FOR SIMULATIONS

Code Used to Simulate α Misbehavior in OPNET

```
/*In Wireless Lan Node process model code
for BKOFF_NEEDED stage is is modified*/
if (backoff_slots == BACKOFF_SLOTS_UNSET)
{
/* Compute backoff interval using binary exponential process*/
/* After a successful transmission we always use cw_min*/
if (short_retry_count + long_retry_count == 0 ||
  wlan_flags->perform_cw == OPC_TRUE)
{
/* If retry count is set to 0 then set the maximum backoff*/
/* slots to min window size*/
max_backoff =cw_min;
}
else
{
/* We are retransmitting. Increase the back-off window*/
/* size.Instead of Two we use alpha as multipliative factor*/
max_backoff = max_backoff*2+1;
}
/* The number of possible slots grows exponentially until it*/
/* exceeds a fixed limit*/
if (max_backoff >cw_max)
{
max_backoff =cw_max;
}
```

```
/* Obtain a uniformly distributed random integer between 0 and*/
/* the minimum contention window size. Scale the number of*/
/* slots according to the number of retransmissions*/
backoff_slots = floor (op_dist_uniform (alpha*(max_backoff+1)));
}
```

Code Used to Simulate β Misbehavior in OPNET

```
if (backoff_slots == BACKOFF_SLOTS_UNSET)
{
/* Compute backoff interval using binary exponential process*/
/* After a successful transmission we always use cw_min*/
if (short_retry_count + long_retry_count == 0
|| wlan_flags->perform_cw == OPC_TRUE)
{
/* If retry count is set to 0 then set the maximum backoff*/
/* slots to min window size*/
if(max_backoff>cw_min)
max_backoff =cw_min;
else  /* to take care of the case where beta < 1*/
max_backoff =(beta*32)-1;
}
else
{
/* We are retransmitting. Increase the back-off window size*/
if(max_backoff<cw_min)
max_backoff=cw_min;
else
max_backoff = (beta*max_backoff+1)-1;
```

```
}
/* The number of possible slots grows exponentially until it*/
/* exceeds a fixed limit*/
if (max_backoff >cw_max)
{
max_backoff =cw_max;
}
/* Obtain a uniformly distributed random integer between 0 and*/
/* the minimum contention window size. Scale the number of*/
/* slots according to the number of retransmissions*/
backoff_slots = floor (op_dist_uniform ((max_backoff+1)));
}
```

Fixed Contention Window (CW_{fix}) Misbehavior

```
if (backoff_slots == BACKOFF_SLOTS_UNSET)
{
/* Compute backoff interval using binary exponential process*/
/* After a successful transmission we always use cw_min*/
if (short_retry_count + long_retry_count == 0
|| wlan_flags->perform_cw == OPC_TRUE)
{
/* If retry count is set to 0 then set the maximum backoff*/
/* slots to min window size*/
max_backoff =cw_min;
}
else
{
```

```
/* We are retransmitting. Increase the back-off window*/
/* size.Instead of Two we use alpha as multipliative factor*/
max_backoff = max_backoff*2+1;

}

/* The number of possible slots grows exponentially until it*/
/* exceeds a fixed limit*/
if (max_backoff >cw_max)
{
max_backoff =cw_max;
}
/* Obtain a uniformly distributed random integer between 0 and*/
/* the minimum contention window size. Scale the number of*/
/* slots according to the number of retransmissions*/
backoff_slots = floor (op_dist_uniform (CW_fix));
/*Backoff values are always chosen from Fixed Contention window*/

}
```

Maximum Contention Window (CW_{max}) Misbehavior

```
if (backoff_slots == BACKOFF_SLOTS_UNSET)
{
/* Compute backoff interval using binary exponential process*/
/* After a successful transmission we always use cw_min*/
if (short_retry_count + long_retry_count == 0 || wlan_flags->
```

```
perform_cw ==OPC_TRUE)
{
/* If retry count is set to 0 then set the maximum backoff*/
/* slots to min window size*/
max_backoff =cw_min;
}
else
{
/* We are retransmitting. Increase the back-off window size*/
max_backoff = max_backoff*2 + 1;
}

/* The number of possible slots grows exponentially until it
*/
/* exceeds a fixed limit.
*/
if (max_backoff >fixed_cw_max)
/*cw_max is always greater than fixed_cw_max*/
{
max_backoff =fixed_cw_max;
}

/* Obtain a uniformly distributed random integer between 0 and */
/* the minimum contention window size. Scale the number of */
/* slots according to the number of retransmissions */
backoff_slots = floor (op_dist_uniform ((max_backoff+1)));
}
```

Deterministic Backoff (b) Misbehavior

```
if (backoff_slots == BACKOFF_SLOTS_UNSET)
{
/* Compute backoff interval using binary exponential process */
/* After a successful transmission we always use cw_min*/
if (short_retry_count + long_retry_count == 0
|| wlan_flags->perform_cw == OPC_TRUE)
{
/* If retry count is set to 0 then set the maximum backoff slots
to min window size*/
max_backoff =cw_min;
}
else
{
/* We are retransmitting. Increase the back-off window size*/
max_backoff = max_backoff*2 + 1;
}

/* The number of possible slots grows exponentially until it
exceeds a fixed limit*/
if (max_backoff >cw_max)
{
max_backoff =cw_max;
}
```

```
/* Obtain a uniformly distributed random integer between 0 and*/
/* The minimum contention window size. Scale the number of*/
/* Slots according to the number of retransmissions*/

backoff_slots = b;
/*b is the deterministic backoff value*/
}
```

NS-2 Code

```
GLOBAL VARIABLES

MAC-802_11.cc

#define node_ID
Address::
instance().get_nodeaddr(((MobileNode*)(netif_->node()))->nodeid())
#define pre_rea_time 200
#define post_rea_time 89800
#define sim_time 90000
#define num_node 10

float pkt_size, tput[10],now,stime_now1[10],
stime_prev1[10],diff1[10],
difftput[10],rea[10],post_conv_tput[10],
post_conv,diff2[nodeid];
int packet_count[10],var,flag = 1,
```

```
packet_threshold[10]=
{100,100,100,100,100,100,100,100,100,100};

MAC-timers.cc

float tempa=0,tempr=0,reatime=0,bebtime=0,t_put[10],
tput_long[10],time[10],time1[10],time2[10],
diff_t[10],tput_genuine =125,delta=10,diff_b[10],simulation_t=900;
int cw_fix[10] = {16,16,16,16,16,16,16,16,16,16},
pkt_r[10] = {100,100,100,100,100,100,100,100,100,100},
count_c[10];
int t[10],l[10],m[10],conv_threshold[10],count_conv = 8,c
ounter[10]={0,0,0,0,0,0,0,0,0,0},
temprcount[10],r[10],tempbcount[10];

/*MAC-802_11.h*/
class Mac802_11 : public Mac
{
void recvpkt_thresh(int p,int id);
void recv_convtime(float c_t);
}
/*MAC-802_11.h*/

/*MAC-timers.h*/
class BackoffTimer : public MacTimer
{
void start
```

```
(int cw, int idle, double difs = 0.0, int nodeid = 0,
int* pkt_count = NULL ,float* diff_ten = NULL);
}
/*MAC-timers.h*/

/*MAC-802_11.cc*/
void
Mac802_11::recvDATA(Packet *p)
{
pkt_size=ch->size();
/*Copy Data size of data packet into temp*/
}



void
Mac802_11::recvpkt_thresh(int p1,int id1)
{
packet_threshold[id1]=p1;
/*Recieving Reaction Packet Threshold from mac timers.cc*/
}
void
Mac802_11::recv_convtime(float conv_time)
{
post_conv = conv_time;
/*Recieving time when convergence happened from mac timers.cc*/
}
```

```
void
Mac802_11::recvACK(Packet *p)
{
packet_count[node_ID]++;
/*Counts the total number of packets successfully sent by each node*/

if(Scheduler::instance().clock()>pre_rea_time)
/*Counts the total number of packets successfully sent by
each node after the reaction is applied*/
rea[node_ID]++;
if((Scheduler::instance().clock())>post_conv&&(post_conv!=0))
/*Counts the total number of packets successfully sent by
each node after convergence happened*/
post_conv_tput[node_ID]++;

now=Scheduler::instance().clock();

if((now>pre_rea_time)&&(flag==1))
{
flag=0;
printf("Pre Reaction Throughput of nodes are \n");
for(var=1;var<10;var++)
printf
("%f \n",packet_count[var]*pkt_size*8/
Scheduler::instance().clock()/1000);
}
```

```
if((now>(sim_time-0.01))&&(flag==0))
{
flag=2;
printf("Post Reaction Throughput of nodes are \n");
for(var=1;var<num_node;var++)
printf("%f\n",rea[var]*pkt_size*8/(post_rea_time*1000));
printf("Cumilative Throughput of nodes are \n");
for(var=1;var<num_node;var++)
printf
("%f\n",packet_count[var]*pkt_size*8/
(Scheduler::instance().clock()*1000));
}

if((now>(sim_time-0.01))&&(flag==2))
{
flag=3;
printf("Post convergence Throughput of nodes are \n");
for(var=1;var<num_node;var++)
printf("%f\n"
,post_conv_tput[var]*pkt_size*8/
((Scheduler::instance().clock()-post_conv)*1000));
}
/*
* Backoff before sending again.
*/
mhBackoff_.start(cw_, is_idle(),0,node_ID,packet_count,diff2);
```

```
/*Send node_ID and Packet Count to
mac timers.cc for choosing backoff*/
done:

}
/*MAC-802_11.cc*/



/*MAC-timers.cc*/
void
BackoffTimer::start
(int cw, int idle, double difs, int nodeid,
 int* pkt_count,float* diff_ten)
{


stime = s.clock();
if((nodeid)>0&&(nodeid<10))
{
if(stime<200)
t_put[nodeid] = pkt_count[nodeid]*pkt_size*8/(stime*1000);
}

if(nodeid==1) /*If the Node is 1,
Alpha Misbehavior is Configured*/
{
```

```
rtime = (Random::random() % int(alpha*cw)) * mac->phymib_.getSlotTime();

}
else if((nodeid)>0&&(nodeid<10))
/*Reaction Approach in a genuine node*/
{

if(stime>200) /*Reaction is applied
only after 200 sec of Simulation
to observe post reaction throughput*/
{
/*Begin of Adaptive Algorithm*/
if(count_c[nodeid]<3)
{
if((l[nodeid]%25 == 0)&&(l[nodeid]!=0))//begin of loop L
{
time[nodeid]=time1[nodeid]+time2[nodeid];
l[nodeid]=0;
m[nodeid]=0;
printf("NODE ID %d*****************
*****************************\n",nodeid);
printf("Tuple being used (%d,%d)
TIME %f\n",pkt_r[nodeid],cw_fix[nodeid],stime);
tput_long[nodeid]= ((pkt_r[nodeid])
*25+m[nodeid]*10)*512*8/(time[nodeid]*1000);

if (tput_long[nodeid] < 93.75)
```

```
/* tput_long[nodeid] < 75% of T_g */
{
pkt_r[nodeid] = pkt_r[nodeid] *2;
cw_fix[nodeid] = cw_fix[nodeid]*0.5;
if(cw_fix[nodeid]<2)
cw_fix[nodeid]=2;
}
/* 2, 0.5 */

else if ((tput_long[nodeid] > 93.75) && (tput_long[nodeid] < 100))
/* tput_long[nodeid] > 75% and tput_long[nodeid] < 80% of T_g */
{
pkt_r[nodeid] = pkt_r[nodeid] *3/2;
cw_fix[nodeid] = cw_fix[nodeid]*2/3;
if(cw_fix[nodeid]<2)
cw_fix[nodeid]=2;
}
/* 1.5,0.67 */

else if ((tput_long[nodeid] > 100) && (tput_long[nodeid] < 112.5))
/* tput_long[nodeid] > 75% and tput_long[nodeid] < 80% of T_g */
{
pkt_r[nodeid] = pkt_r[nodeid] +10;
cw_fix[nodeid] = cw_fix[nodeid] - 1;
if(cw_fix[nodeid]<2)
cw_fix[nodeid]=2;
}
```

```
/*Additive increment decrement*/

else if (tput_long[nodeid] > 125 + delta)
/* delta = 10 */ /* tput_long[nodeid] > 100% of T_g */
{
pkt_r[nodeid] = pkt_r[nodeid] - 20;
cw_fix[nodeid] = cw_fix[nodeid] +1;
}
/*Additive increment decrement*/
/*Else leave the tuple unchanged*/

mac->recvpkt_thresh(pkt_r[nodeid],nodeid);
/*Send the new Packet threshold to MAC-802_11.cc*/

if(conv_threshold[nodeid]==pkt_r[nodeid])
{
count_c[nodeid]++;
/*Update counter whenever a node chooses the same tuple*/
}
else
{
conv_threshold[nodeid]=pkt_r[nodeid];
count_c[nodeid]=0;
/*Reset Counter if node chooses different tuple*/
}
time1[nodeid]=0;
time2[nodeid]=0;
```

```
/*Reset time counters after each decision instance*/
conv_prev[nodeid]=stime;
}
}
else
/*If Counter is greater than three*/
{
if(counter[nodeid]==0)
{
count_conv--;
/*Number of nodes - count_conv gives the
number of nodes converged*/
if(count_conv==0)
/*If all nodes are converged*/
{
printf("convergence happened at time %f",stime);
mac->recv_convtime(stime);
/*Declare convergence and send time at which
convergence happened to
MAC-802_11.cc*/
count_conv--;
}
printf("NODE %d Converged with Tuple(%d,%d)
and Total converged nodes
are %d\n"
,nodeid,int((pkt_r[nodeid]/pkt_b)*pkt_b),
cw_fix[nodeid],(8-count_conv));
```

```
counter[nodeid]=1;
/*To make sure each node converges only once*/
}
}
/*End of Adaptive Algorithm*/

if(t_put[nodeid]<threshold)
/*If Throughput is less than the specified
threshold employ the reaction*/
{

rtime = (Random::random() % int(cw_fix[nodeid]))
 * mac->phymib_.getSlotTime();
/*Node employing CW_fix Misbehavior as a
reaction approach*/

if(nodeid==3)
/*To compute the percentage amount of time backoff is
employed aggressively in the reaction*/
{
tempr++;
reatime = reatime + rtime;
}


if
((pkt_count[nodeid]!=0)&&(pkt_count[nodeid]!=
```

```
temprcount[nodeid])&&(pkt_count[nodeid]!=tempbcount[nodeid]))
/*To make sure that for retransmission attempts do
not count towards the Successful attempts*/
{
temprcount[nodeid] = pkt_count[nodeid];
if(r[nodeid]==0)
{
diff_t[nodeid]=stime;
/*Computing time taken to send pkt_r number of packets*/
}
r[nodeid]++;
/*Counting pkt_r number of packets*/
if(r[nodeid]==(pkt_r[nodeid]))
{
diff_t[nodeid]=stime-diff_t[nodeid];
t_put[nodeid]= pkt_r[nodeid]*pkt_size*8/(diff_t[nodeid]*1000);
/*Computing throughput over time pkt_r number of packets are sent*/
l[nodeid]++;
/*Rea_interval*/
time1[nodeid]=diff_t[nodeid]+time1[nodeid];
/*Computing time taken in Reacting to send
Rea_Interval*pkt_r number of packets*/
r[nodeid]=0;
/*Resetting counter after each increment in Rea_interval*/
diff_t[nodeid]=0;
}
}
```

```
}

else
/*If Throughput is not less than the specified threshold employ BEB*/
{

rtime = (Random::random() % cw) * mac->phymib_.getSlotTime();
/*Employing BEB*/

if(nodeid==3)
/*To compute the percentage amount of time b
ackoff is employed using BEB in the reaction.*/
{
tempa++;
bebtime = bebtime + rtime;
}
if((pkt_count[nodeid]!=0)&&(pkt_count[nodeid]!=
tempbcount[nodeid])&&(pkt_count[nodeid]!=temprcount[nodeid]))
/*To make sure that for retransmission attempts
do not count towards the Successful attempts*/
{
tempbcount[nodeid] = pkt_count[nodeid];
if(t[nodeid]==0)
{
diff_b[nodeid]=stime;
/*Computing time taken to send pkt_b number of packets*/
```

```
}
t[nodeid]++;
/*Counting pkt_r number of packets*/
if(t[nodeid]==pkt_b)
{
diff_b[nodeid]=stime-diff_b[nodeid];
time2[nodeid]=diff_b[nodeid]+time2[nodeid];
/*computing time taken in BEB to
send Rea_Interval*pkt_r number of packets*/
m[nodeid]++;
/*To compute Number of packets sent
Employing BEB during Rea_interval L*/
t_put[nodeid]= pkt_b*pkt_size*8/(diff_ten[nodeid]*1000);
/*Computing throughput over time
pkt_r number of packets are sent*/
t[nodeid]=0;
/*Resetting counter after each increment in m*/
diff_b[nodeid]=0;
}
}

}
}

else
/*For time less than 200 sec Employ BEB to
capture pre reaction throughput of the nodes*/
```

```
{
rtime = (Random::random() % cw) * mac->phymib_.getSlotTime();
}

if((stime>simulation_t-0.01))
/*Print the percentage amount of times backoff
is employed using BEB and Aggressively in the reaction*/
{
printf("Fraction of aggressive transmission attempts as a
result of reaction scheme is %lf fraction of BEB attempts is %lf\n",
(tempr/(tempa+tempr))*100,(tempa/(tempa+tempr))*100);
printf("Genuine nodes are aggressively backing off %lf
fraction of time and BEB is used is %lf fraction of time\n",
(reatime/(reatime+bebtime))*100,(bebtime/(reatime+bebtime))*100);
}

}
}
/*MAC-timers.cc*/
```

TCL Code

```
# Initialize some parameters
Mac/802_11 set dataRate_ 2Mb
# rts thrshold is in bytes
Mac/802_11 set RTSThreshold_ 128

# ======================================================================
# Define options
```

```
# ======================================================================
# simulation time in sec
set sim(end) 4000

set val(chan)    Channel/WirelessChannel  ;# channel type
set val(prop)    Propagation/TwoRayGround ;# radio-propagation model
set val(netif)   Phy/WirelessPhy          ;# network interface type
set val(mac)     Mac/802_11               ;# MAC type
set val(ifq)     Queue/DropTail/PriQueue  ;# interface queue type
set val(ll)      LL                       ;# link layer type
set val(ant)     Antenna/OmniAntenna      ;# antenna model
set val(ifqlen)  50                       ;# max packet in ifq
set val(nn)      10                       ;# number of mobilenodes
set val(rp)      DumbAgent                ;# routing protocol

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

# Initialize Global Variables
#
set ns_         [new Simulator]

#Open the NAM trace file

set tracefd     [open wireless.tr w]
#set namfd       [open wireless.nam w]
```

```
set f0          [open wireless.data w]
set f1      [open wireless1.data w]
set nodeid      [open nodeid.data w]

$ns_ trace-all $tracefd

# grid values X and Y are in meters
#$ns_ namtrace-all-wireless $namfd 100 100

# set up topography object
set topo       [new Topography]

$topo load_flatgrid 100 100
#
# Create God
#
create-god $val(nn)

#
#  Create the specified number of mobilenodes [$val(nn)]
and "attach" them
#  to the channel.
#  Here ten nodes are created-nodes 0 through 9

# configure node
#
        $ns_ node-config -adhocRouting $val(rp) \
```

```
 -llType $val(ll) \
 -macType $val(mac) \
 -ifqType $val(ifq) \
 -ifqLen $val(ifqlen) \
 -antType $val(ant) \
 -propType $val(prop) \
 -phyType $val(netif) \
 -channelType $val(chan) \
 -topoInstance $topo \
 -agentTrace OFF \
 -routerTrace OFF \
 -macTrace OFF \
 -movementTrace OFF

for {set i 0} {$i < $val(nn) } {incr i} {
set node_($i) [$ns_ node]
$node_($i) random-motion 0;# disable random motion
}
#$node_(1) color red
#
# Provide initial (X,Y, for now Z=0) co-ordinates for mobilenodes
#
$node_(0) set X_ 50.0
$node_(0) set Y_ 50.0
$node_(0) set Z_ 0.0

##nodes in a circular way
```

```
set pi 3.1416
set r  30
set a [expr {2*$pi*0.111111111}]

for {set i 1} {$i < $val(nn) } {incr i} {

$node_($i) set X_ [ expr { 50 + ($r * cos($i * $a))}]
$node_($i) set Y_ [ expr { 50 + ($r * sin($i * $a))}]
$node_($i) set Z_ 0.0

$ns_ initial_node_pos $node_($i) 2
}
$ns_ initial_node_pos $node_(0) 2


#traffic between nodes
for {set i 1} {$i < $val(nn) } {incr i} {
set udp($i) [new Agent/UDP]
$udp($i) set class_ 2

set cbr($i) [new Application/Traffic/CBR]
$cbr($i) attach-agent $udp($i)
#100 packets per second
$cbr($i) set packetSize_ 512
$cbr($i) set interval_ 0.01
#cbr($i) set burst_time_ 3600
#cbr($i) set idle_time_ 1
```

```
$ns_ attach-agent $node_($i) $udp($i)

set sink($i) [new Agent/LossMonitor]
#set sink($i) [new Agent/Null]
$ns_ attach-agent $node_(0) $sink($i)
$ns_ connect $udp($i) $sink($i)


$ns_ at 0.0 "$cbr($i) start"

}

#
# Tell nodes when the simulation ends
#
for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at $sim(end) "$node_($i) reset";
}

$ns_ at $sim(end) "puts \"NS EXITING...\" "
$ns_ at $sim(end) "stop"
proc stop {} {
    global ns_ tracefd namfd f0
    $ns_ flush-trace
    #close $namfd
    close $tracefd
  #exec nam wireless.nam &
```

```
    $ns_ halt
}

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