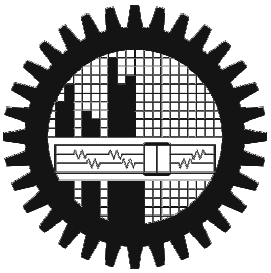

BANGLADESH UNIVERSITY OF ENGINEERING & TECHNOLOGY

Tunable Parameters for IEEE 802.11 based Ad-Hoc Network

by

S.M. Rifat Ahsan

Student No. 0405023

Mohammad Saiful Islam

Student No. 0405010

Naeemul Hassan

Student No. 0405018

A thesis submitted in partial fulfillment for the
degree of Bachelor of Science in Computer Science & Engineering

in the

Faculty of Electrical & Electronic Engineering
Department of Computer Science & Engineering

October 2009

Declaration of Authorship

We, declare that this thesis titled, "Tunable Parameters for IEEE 802.11 based Ad-Hoc Network' and the work presented in it are our own. We confirm that:

- This work was done wholly or mainly while in candidature for the degree of Bachelor of Science in Computer Science & Engineering at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where We have consulted the published work of others, this is always clearly attributed.
- Where We have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- We have acknowledged all main sources of help.

S.M. Rifat Ahsan
Student No. 0405023

Mohammad Saiful Islam
Student No. 0405010

Naeemul Hassan
Student No. 0405018

Certificate

The thesis titled "Tunable Parameters for IEEE 802.11 based Ad-Hoc Network" submitted by S.M. Rifat Ahsan, Mohammad Saiful Islam, and Naeemul Hassan, has been accepted as satisfactory in partial fulfillment of the requirement for the degree of Bachelor of Science & Engineering in Computer Science & Engineering held in October 2009.

Dr. A.K.M Ashikur Rahman
Assistant Professor
Computer Science & Engineering
Bangladesh University Of Engineering & Technology
Dhaka-1000, Bangladesh

"If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization."

Murphy's Laws of Technology

Abstract

IEEE 802.11 is a well known standard for wireless local area networks for both Infrastructure Based and Ad-Hoc networks. This standard uses several parameters which administrates the operation of the protocol in Physical and Medium Access Control(MAC) layer of the network. In our thesis research we have worked on various parameters for the MAC layers. In the original standard these parameters have fixed values. But our work began by searching for procedures to tune the parameters in adaptive ways. Later we have concentrated our search on a particular parameter; RTS Threshold. IEEE 802.11 Medium Access Control (MAC) protocol employs two techniques for packet transmission; the basic access scheme and the RTS/CTS reservation scheme. RTS Threshold plays an important role in the performance of the network by choosing a particular scheme to use for a transmission. In our research, we, point out the advantages and disadvantages of RTS/CTS scheme. Then we state the problems of having a fixed RTS Threshold. Next, we present a numerical method to fix RTS threshold adaptively based on network traffic to make a balance between the basic Scheme and the RTS/CTS scheme for having a optimized network throughput. All theoretical analysis and algorithms are validated by simulation. We have used the Network Simulator (NS2) tool for the experimental simulations. We substantiate that our proposal outperforms the IEEE 802.11 standard in the static network situation and also in multi hop environment considering hidden node problem.

Acknowledgements

First of all we would like to thank our supervisor, Dr. A.K.M Ashikur Rahman, for introducing us to the amazingly interesting world of Ad-Hoc Wireless Network and teaching us how to perform research work. We, ourselves were interested about wireless networking but he had shown us the great treasure of this ever flourishing field of communications. He taught us how to go through a previous research paper, analyze a problem and how to conduct simulation in NS2. Without his continuous supervision, guidance and valuable advice, it would have been impossible to complete the thesis. We are especially grateful to him for his encouragement at times of disappointment, and for his patience with our wildly sporadic work habits. We are grateful to all other friends for their continuous encouragement and for helping us in thesis writing. We would like to express our gratitude to all our teachers. Their motivation and encouragement in addition to the education they provided meant a lot to us. We have taken support from various sources about Ad-Hoc Wireless Networks, NS2, Latex and we are thankfull to the writers of those books, publishers of the websites. Last but not least, we are grateful to our parents and to our families for their patience, interest, and support during our studies.

Contents

List of Figures

List of Tables

Dedicated to Our parents -

S.M Rifat Ahsan

Mohammad Saiful Islam

Naeemul Hassan

Chapter 1

Introduction

1.1 Communication and Networks

A computer network is an interconnected collection of independent computers which aids communication in numerous ways. Apart from providing good communication medium, sharing of available resources, improved reliability of service, and cost-effectiveness are some of the advantages of computer networking.

A computer network allows computers to communicate with many other and to share resources and information. The Advanced Research Projects Agency (ARPA) funded the design of the "Advanced Research Projects Agency Network" (ARPANET) for the United States Department of Defense. It was the first operational computer network in the world. Development of the network began in 1969, based on designs starting in the 1960s.

1.1.1 Wireless Network

Wireless network refers to any type of computer network that is wireless, and is commonly associated with a telecommunications network whose interconnections between nodes is implemented without the use of wires. Wireless telecommunications networks are generally implemented with some type of remote information transmission system that uses electromagnetic waves, such as radio waves, for the carrier and this implementation usually takes place at the physical level or "layer" of the network.

Types of wireless networks:

- Wireless Personal Area Network (WPAN) is a type of wireless network that interconnects devices within a relatively small area, generally within reach of a person. For example, Bluetooth.
- Wireless Local Area Network (WLAN) is a wireless alternative to a computer Local Area Network (LAN) that uses radio instead of wires to transmit data back and forth between computers in a small area such as a home, office, or school. Wireless LANs are standardized under the IEEE 802.11 series. For example, Wi-Fi.
- Wireless Metropolitan area networks are a type of wireless network that connects several Wireless LANs. For example, WiMAX.

1.1.2 Wirelss Ad-Hoc Network

In the last few years, there has been a big interest in Ad-Hoc Wireless Networks as they have tremendous military and commercial potential. An ad-hoc wireless network is a wireless network, comprised of mobile computing devices that use wireless transmission for communication, having no fixed infrastructure(a central administration such as base station in cellular network or an access point in wireless local network.) The mobile devices also serves as routers due to the limited range of the wireless transmission of these devices, that is, several devices may need to route or relay a packet before it reaches to its final destination. Ad hoc wireless network can be deployed quickly anywhere and anytime as they eliminate the complexity of infrastructure setup.

The decentralized nature of wireless ad hoc networks makes them suitable for a variety of applications where central nodes can't be relied on, and may improve the scalability of wireless ad hoc networks compared to wireless managed networks, though theoretical and practical limits to the overall capacity of such networks have been identified.

Minimal configuration and quick deployment make ad hoc networks suitable for emergency situations like natural disasters or military conflicts. The presence of a dynamic and adaptive routing protocol will enable ad hoc networks to be formed quickly.

Wireless ad hoc networks can be further classified by their application:

- mobile ad hoc networks (MANETs)
- wireless mesh networks
- wireless sensor networks.

1.2 Motivation and Our Contribution

IEEE 802.11 standard uses a numerous parameters for its operation. In the standard these parameters are given default values. Some parameter values can be set by the user explicitly. Work had been done by various researchers for tuning of these parameters. They had been trying to set the values adaptively on the basis of some kind of network performance matrix. The objective of setting the parameter values adaptively is to maximize the gain(maximizing throughput, minimizing delay or access time, minimizing collision, minimizing power consumption etc.)

Till now most wireless networks are logically subordinate to existing wired networks. IEEE 802.11 was designed to complement existing LANs, not replace them. However, networks have a way of growing, and users have a way of becoming more demanding. Network's performance "out of the box" is probably fairly poor, even if no one notices. Changing the physical environment (by experimenting with access point placement, external antennas, etc.) may alleviate some problems, but others may best be resolved by tuning administrative parameters.

Some of the parameters used in the IEEE 802.11 MAC layer are; BeaconInterval, RTS-Threshold, Fragmentation-Threshold, Long Retry Limit, Short Retry Limit, Listen interval, DTIM Window, ATIM Window, Active Scan Timer, Passive Scan Timer, AuthenticationTimeout, Association Timeout. But only RTS-Threshold, Long Retry Limit, Short Retry limit, Fragmentation Threshold, and Contention Window Size are used by DCF mode of operation.

In our thesis we have investigated RTS-Threshold, Retry Limits, and Contension Window Size for tuning them. We looked into the past works related to the adaptive tuning of these parameters. Finally we have chosen RTS-Threshold for our work.

RTS-Threshold is an important parameter in DCF mode of operation. The value of RTS-Threshold can effect available radio capacity, throughput, and battry life. But setting the value arbiterily can effect the throughput badly.

We have found that there is much space for research on tuning of RTS-Threshold. After a long research and experiments we have finally devised a method of adaptively tuning the RTS threshold value with the help of current packet size distribution of the network. Here this distribution means the size of the packets currently flowing through the network.

Chapter 2

Background

2.1 Ad Hoc Wireless Network

Multi hop-relaying is the principle behind Ad-Hoc Networks. King Darius I, in about 500 BC started using this technique, for transmitting messages around his ruled areas. He placed men, on top of tall structures, who would relay the message from one person to another. Researchers from University of Hawaii, in 1970 invented ALOHAnet, which was used to link various universities on the Hawaiian islands. ALOHAnet was the first networks to use common medium sharing among various stations. The success of the system triggered the work of packet radio network(PRNET) project sponsored by defense advanced research project agency(DARPA)[1]. hough the initial attempt had a centralized control, it quickly evolved into a distributed multi hop wireless communication system that could operate over a large area. PRNET used ALOHA and carrier sense multiple access(CSMA) to share common resource, which was a better technology than basic ALOHAnet scheme. The system was designed to self-organize, self-configure, and a direct radio connectivity for the dynamic operation of a routing protocol without any support from fixed architecture.

Realizing the necessity of open standards in this emerging area of communication network Internet Engineering Task Force(IETF), termed the mobile ad hoc networks(MANET) working group [2], was formed to standardize the protocols and functional specifications of wireless ad hoc networks. Recent advances in wireless network architectures reveal solutions that enable the ad hoc network to work in the presence of infrastructure. Multi hop cellular networks(MCN)[3], and self organizing packet radio ad hoc networks with overlay(SOPRANO) [4] are examples of such type. These hybrid networks increases the capacity of system significantly.

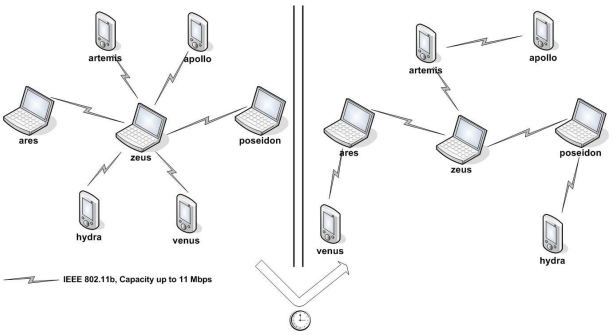

FIGURE 2.1: Ad-Hoc Wireless Network

2.1.1 Cellular and Ad Hoc Wireless Networks

The current cellular networks are classified as the infrastructure dependent networks. The path setup between two nodes is completed through the base station.

Ad hoc wireless networks are capable of operating without the support of any fixed infrastructure. The absence of any central control system makes the routing complex compared to cellular networks. The path setup between two nodes in ad hoc network is done through intermediate nodes. For the distributive system to work the mobile nodes of ad hoc network are needed to be more complex than that of cellular networks.

2.1.2 Application of Ad Hoc Wireless Networks

Military Applications

Setting up fixed infrastructure in enemy territory for communications among group of soldiers may not be possible. In such environment ad hoc networks provide required communication mechanism quickly. Another application in this area can be the coordination of military objects moving at a high speed.

Collaborative and Distributive Computing

The requirement of a temporary communication infrastructure for quick communication with minimal configuration among a group of people in a meeting or conference necessitates the formation of an ad hoc wireless network.

Emergency Operations

Ad hoc networks are very useful in emergency operations such as search and rescue, and crowd control. The major factor that favors ad hoc networks for such task is the self configuration of the system with minimal overhead, the freedom of flexibility and mobility, and the unavailability of infrastructure. In many situations like war or natural disaster, the infrastructure can be destroyed and ad hoc networks can be deployed quickly.

Wireless Mesh Networks

Wireless mesh networks are ad hoc wireless networks that are formed to provide an alternative communication infrastructure for mobile or fixed nodes, without the spectrum reuse constraints and the requirements of network planning of cellular networks. The investment required in the system is much less than what is required for the cellular networks. Such networks are formed by placing wireless relaying equipment spread across the area to be covered by the network. The possible deployment scenarios are, residential zones, highways, business zones, and university campuses.

Wireless Sensor Networks

Sensor networks are special types of ad hoc networks that are used to provide wireless communication infrastructure among the sensors deployed in a specific application domain. Sensor nodes are tiny devices that have the capability of sensing physical parameters, processing the data gathered, and communicating over the network to the monitoring station. The issues that differs it from a traditional ad hoc are,

- Non mobility of nodes
- Large size of network
- Varying density of networks.
- Power Constraints.
- Data fusion
- Varying traffic distribution.

Hybrid Wireless Network

This is the mixture of ad hoc and infrastructure based wireless networks. The major advantages of this type of network is,

- Higher capacity than cellular networks.
- Increased flexibility and reliability in routings.
- Better coverage and connectivity in holes(areas that are not covered by fixed infrastructure).

2.1.3 Issues in Wireless Ad Hoc Networks

The major issues that are discussed in wireless ad hoc networks are,

- Medium Access Scheme
- Routing
- Multicasting
- Transport layer protocol
- Pricing scheme
- QoS provisioning
- Self-organization
- Security
- Energy management
- Addressing and service discovery
- Scalability
- Deployment considerations

2.2 Medium Access Control Protocols for Ad Hoc Wireless Networks

Nodes in the wireless network share a common broadcast radio channel. Since the radio spectrum is limited, the bandwidth available for communication in such networks is also limited. Access to this shared medium should be controlled in such manner that all nodes receive a fair share of the available bandwidth, and the bandwidth is utilized efficiently.

2.2.1 Issues in Designing MAC Protocol for Ad Hoc Wireless Network

Bandwidth Efficiency

The MAC protocol must be designed in such a way that the scarce bandwidth is utilized in an efficient manner. The control overhead must be kept as minimal as possible. Bandwidth efficiency can be defined as ratio of bandwidth used for actual data transmission to the total available bandwidth. MAC protocol must maximize bandwidth efficiency.

Quality of Service Support

Due to the inherent nature of the nodes in the system, providing QoS support to data sessions is very difficult. Bandwidth reserved at the start of transmission may become invalid due to mobility of nodes. QoS support is necessary for time critical traffic sessions. MAC protocol that supports real time traffic must support some kind of resource reservation mechanism that take into consideration mobility of nodes.

Synchronization

The MAC protocol must take into consideration the synchronization between nodes in the network. Synchronization is necessary for bandwidth reservation. Exchange of control packets may be required for achieving time synchronization among nodes.The control packets must not consume too much bandwidth

Hidden and Exposed Terminal Problems

The hidden and exposed terminal problems are unique to wireless networks. The hidden and exposed terminal problems significantly reduce the throughput of a network when

the traffic load is high. It is therefor desirable that the MAC protocol be free from the hidden and exposed terminal problems

Error Prone Shared Broadcast Channel

When a node is receiving no other nodes apart from the sender should transmit. A node should get access to medium, if its transmission does not interfere any ongoing session. The collision probability is very high in the wireless network. A MAC protocol must minimize collision.

Lack of Central Coordination

In cellular network the base station allocate bandwidth to the mobile nodes, but this is not possible for ad hoc networks. Therefore the nodes should be scheduled in a distributive manner for gaining access to the medium.This may require exchange of control information. The MAC protocol must ensure that the bandwidth overhead of this exchange is not too high.

Mobility of Nodes

Nodes are mobile most of the times. If mobility is too high then bandwidth reservation or channel reservation may become useless The protocol must ensure that performance does not effect to much due to mobility.

2.2.2 Design Goals of a MAC Protocol for Wireless Ad Hoc Network

- The operation should be distributed.
- Should support QoS for real time traffic.
- The access delay must be kept low.
- Bandwidth must be used efficiently.
- Fair allocation of resource must be ensured.
- Control overhead must be kept minimum.
- Should minimize the effect of hidden and exposed node problems.
- Must be scalable to large networks.

- It should have power control mechanisms in order to efficiently manage energy consumptions of nodes.
- The protocol should have mechanisms for adaptive data rate control.
- Should provide time synchronization among nodes.

2.2.3 Classifications of MAC protocols

MAC protocols for ad hoc wireless networks can be classified into several categories based on various criteria such as initiation approach, time synchronization, and reservation approach. Ad hoc network MAC protocols can be classified into three basic types,

- Contention based protocols.
- Contention based protocols with reservation mechanism.
- Contention based protocols with scheduling mechanism.

Apart from these, there exist other MAC protocols that can not be classified clearly under any one of the above three types.

Contention Based Protocols

These protocols follow a contention based channel access policy. A node does not make any resource reservation in priori. Whenever it receives a packet to be transmitted, it contends with other nodes for access to the shared channel. This system can not provide QoS guarantee to session since nodes are not guaranteed regular access to the channel. They are further divided in two types,

- Sender initiated protocols. Packet transmission are initiated by sender node.
- Receiver initiated protocols. The receiver node initiates the contention resolution protocol.

Sender initiated protocols are further divided into two types,

- Single channel sender initiated protocols. Here the total available bandwidth is used as it is, without being divided. A node that wins the contention to the channel can make use of the entire bandwidth.

- Multichannel sender initiated protocols. Here the available bandwidth is divided into multiple channels. This enables several nodes to simultaneously transmit data, each using a separate channel. Some protocols dedicate a frequency channel exclusively for transmitting control information.

Contention Based Protocols with Reservation Mechanisms

Ad hoc wireless networks sometimes may need to support real time traffic, which requires QoS guarantees to be provided. In order to support such traffic, certain protocols have mechanisms for reserving bandwidth in priori. Such protocols can guarantee QoS to time sensitive traffic sessions. These protocols are classified into two types,

- Synchronous protocols: These systems require time synchronization among all the nodes in the network, so that reservation made by a node are known to other nodes in its neigbourhood. Global time synchronization is difficult to achieve.
- Asynchronous protocols: They do not require any global synchronization among the nodes. These protocols usually use relative time information for effecting reservations.

Contention Based Protocols with Scheduling Mechanisms

These protocols focus on packet scheduling at nodes, and also scheduling nodes for access to the channel. Node scheduling is done in a manner so that all nodes are treated fairly. Scheduling based schemes are also used for enforcing priorities among flows whose packets are queued at nodes. Some scheduling schemes also take into consideration battery characteristics, such as remaining battery power, while scheduling nodes for access to the channel.

2.3 IEEE 802.11 Standard

The IEEE 802.11 standard specifies the physical layer, MAC layer, adapted to the specific requirements of wireless LANs. The objective of this standard is to provide wireless connectivity to devices that requires rapid deployment. This standard was brought out in 1997. A later version is IEEE 802.11b [3], commercially known as Wi-Fi. The 802.11 working group had to examine connection management, link reliability management, and power management. In addition provision for security had to be introduced.

2.3.1 802.11 MAC Layer

802.11 standard supports two modes of operations. The first is called DCF(Distributed Coordination Function), which is an ad hoc mode and does not require any central control. The second is called PCF(Point Coordination Function), uses the base station to control all the activities in its cell. It is the infrastructure mode.

The basic service supported are the mandatory asynchronous data service and an optional real-time service. The asynchronous data service is supported for both unicast and multi cast packets. The real time service is supported only in infrastructure based networks.

Inter-Frame Spacing

Inter frame spacing refers to the time interval between the transmission of two successive frames by any station. There are four types of IFS, SIFS, PIFS, DIFS,and EIFS, in order from shortest to longest. They denote priority levels of access to medium. Shorter IFS denotes higher priority to access the medium. The exact values of IFS are obtained from the attributes specified in the physical layer management information base(PHYMIB) and are independent of the station bit rate.

- **Short IFS(SIFS)** is defined for short control messages such as acknowledgments for data packets and polling responses. The transmission of any packet should begin only after the channel is sensed to be idle for a minimum time period of at least SIFS.
- **PCF IFS(PIFS)** is the waiting time used for real time service.
- **DCF IFS(DIFS)** is for asynchronous data transfer within contention period.
- **Extended IFS(EIFS)** is used for resynchronizations whenever physical layer detects incorrect MAC frame reception.

RTS-Threshold

A node wishing to send data initiates the process by sending a Request to Send frame (RTS). The destination node replies with a Clear To Send frame (CTS). Any other node receiving the RTS or CTS frame should refrain from sending data for a given time (solving the hidden node problem). The amount of time the node should wait before trying to get access to the medium is included in both the RTS and the CTS frame. This

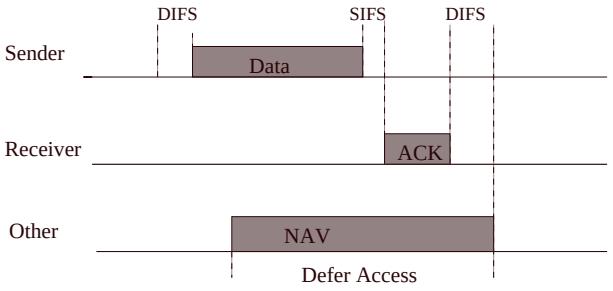

FIGURE 2.2: Basic CSMA/CA Access Mechanism

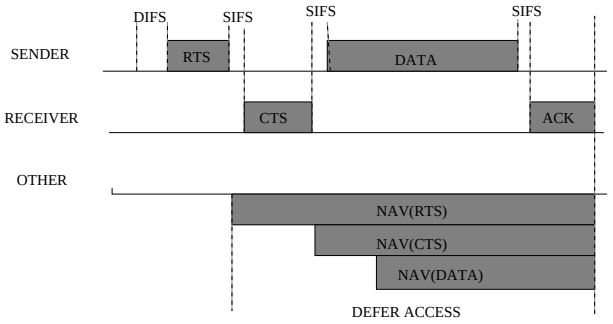

FIGURE 2.3: CSMA/CA With RTS/CTS Mechanism

protocol was designed under the assumption that all nodes have the same transmission range. RTS/CTS is an additional method to implement virtual carrier sensing in Carrier sense multiple access with collision avoidance (CSMA/CA). By default, 802.11 relies on physical carrier sensing only which is known to suffer from the hidden terminal problem.

RTS/CTS protocol description

The RTS Threshold (RT) value which determines when the RTS/CTS handshaking mechanism should be used is an important parameter to investigate; since different RTS-Threshold values will produce different performance characteristics in data transmission.If the packet size the node wants to transmit is larger than the threshold, the RTS/CTS handshake gets triggered. If the packet size is equal to or less than threshold the data frame gets sent immediately. Using a small value causes RTS packets to be sent more often, consuming more of the available bandwidth, therefore reducing the apparent throughput of the network packet. However, the more RTS packets that are sent, the quicker the system can recover from interference or collisions – as would be the case in a heavily loaded network, or a wireless network with much electromagnetic interference.

DCF Medium Access Mechanism

The DCF is based on CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) where carrier sensing is done by physical sensing and virtual sensing. Both sensing mechanisms are used to determine the state of the medium. For physical carrier sensing traditional CSMA/CA, as shown in Fig: 2.2 is used. It requires the nodes to first sense the channel to check whether it is idle for a DCF Inter-frame Space (DIFS) interval, then attempts packet transmission. On the other hand, for virtual carrier sensing(VCS), RTS/CTS handshake and Network Allocation Vector(NAV) scheme is used as shown in Fig: 2.3. The Virtual Carrier Sensing employs RTS/CTS packets exchange for channel reservation. The sender transmits a Request-To-Send frame to it's receiver. The receiver sends a Clear-To-Send frame if the NAV at the receiver indicates idle channel. Then the sender transmits the DATA frame and waits for acknowledgment ACK.

2.4 Tunable Parameters of IEEE 802.11

Here is a list of administrative parameters that are used in IEEE 802.11 standard. These parameters play very important roles in the operation of a network. In the original standard fixed values are used for these parameters. But performance of a network can be improved in many ways by tuning these parameters in optimum values.

Table 2.1: Tunable Parameters of 802.11

Parameter name	Meaning and units	Effect when decreased	Effect when increased
Beacon Interval	Number of TUs between transmission of Beacon frames.	Passive scans complete more quickly, and mobile stations may be able to move more rapidly while maintaining network connectivity	Small increase in available radio capacity and throughput and increased battery life.
RTS Threshold	Frames larger than the threshold are preceded by RTS/CTS exchange.	Greater effective throughput if there are a large number of hidden node situations .	Maximum theoretical throughput is increased, but an improvement will be realized only if there is no interference.

Fragmentation Threshold	Frames larger than the threshold are transmitted using the fragmentation procedure.	Interference corrupts only fragments, not whole frames, so effective throughput may increase.	Increases throughput in noise-free areas by reducing fragmentation acknowledgment overhead.
Long Retry Limit	Number of retransmission attempts for frames longer than the RTS threshold.	Frames are discarded more quickly, so buffer space requirement is lower.	Retransmitting up to the limit takes longer and may cause TCP to throttle back on the data rate.
Short Retry Limit	Number of retransmission attempts for frames shorter than the RTS threshold.	Same as long retry limit	Same as long retry limit.
Listen Interval	Number of Beacon intervals between awakenings of power-saving stations.	Latency of unicast frames to station is reduced. Also reduces buffer load on access points.	Power savings are increased by keeping transceiver powered off for a larger fraction of the time.
DTIM Window	Number of Beacon intervals between DTIM transmissions (applies only to infrastructure networks).	Latency of multi cast and broadcast data to power-saving stations is reduced. Also reduces buffer load on access points.	Power savings are increased by keeping transceiver powered off for a larger fraction of the time.
ATIM Window	Amount of time each station remains awake after a Beacon transmission in an independent network.	Increases power savings by allowing mobile stations to power down more quickly after Beacon transmission.	Latency to power-saving stations is reduced, and the buffer load may be decreased for other stations in the network.
Active Scan Timer	Amount of time a station waits after sending a Probe Response frame to receive a response.	Station moves quickly in its scan.	Scan takes longer but is more likely to succeed.

Passive Scan Timer	Amount of time a station monitors a channel looking for a signal.	Station may not find the intended network if the scan is too short.	Scan takes longer but is more likely to succeed.
Authentication Timeout	Maximum amount of time between successive frames in authentication sequence.	Authentications must proceed faster; if the timeout is too low, there may be more retries.	No significant effect.
Association Timeout	Maximum amount of time between successive frames in association sequence.	Associations must proceed faster; if the timeout is too low, there may be more retries.	No significant effect.

2.5 Past Works

In the past few years wireless network has developed much and many works has been done to increase its performance. Most of them has been done on the basis of 2-D Markov Chain model. An intuitive mathematical analysis and simple equations were presented for throughput and packet delay performance of IEEE 802.11 DCF by utilizing a Markov chain model by [5] which is presented below.

Markov Chain Model Assumptions

Mathematical Modeling Assumptions

- Packets can encounter collisions only due to simultaneous transmissions (no transmission errors)
- There are no hidden stations (all stations can hear others transmissions).
- The network consists of a finite number of contending stations.
- Saturated conditions, i.e. a station has always data ready for transmission.
- The collision probability of a transmitted packet is constant and independent of the number of retransmissions.

Analytical Model

Utilizing a Markov chain model and after some algebra, the probability τ that a station transmits in a randomly chosen slot equal to:

$$if m \leq m' \; \tau = \frac{2(1-2p)(1-p^{m+1})}{W(1-(2p)^{m+1})(1-p)+(1-2p)(1-p^{m+1})} \tag{2.1}$$

$$if m > m' \; \tau = \frac{2(1-2p)(1-p^{m+1})}{W(1-(2p)^{m'+1})(1-p)+(1-2p)(1-p^{m+1})+W2^{m'}p^{m'+1}(1-2p)(1-p^{m-m'})} \tag{2.2}$$

where m is the retry limit, m' identifies the maximum number of backoff stages, W is the contention window (CW) size and p is the packet collision probability given by:

$$p = 1-(1-\tau)^{n-1} \tag{2.3}$$

Time Interval Durations

The values of Ts and Tc depend on the medium access scheme and for the basic access are given by:

$$T_s^{bas} = T_c^{bas} = DIFS + T_{header} + \frac{l}{C} + SIFS + T_{ACK} \tag{2.4}$$

and for the RTS/CTS scheme:

$$T_s^{RTS} = DIFS + T_{RTS} + SIFS + T_{CTS} + SIFS + T_{header} + \frac{l}{C} + SIFS + T_{ACK} \tag{2.5}$$

$$T_c^{RTS} = DIFS + T_{RTS} + SIFS + T_{CTS} \tag{2.6}$$

where l is the payload length, C is the data rate, $C_{control}$ is the control rate, $T_{header}, T_{ACK}, T_{RTS}$ and T_{CTS} are the time intervals required to transmit the packet payload header, the ACK, RTS and CTS control packets, respectively.

$T_{header} = \frac{MAC_{hdr}}{C} + \frac{PHY_{hdr}}{C_{control}}$ $T_{ACK} = \frac{I_{ACK}}{C_{control}}$
$T_{RTS} = \frac{I_{RTS}}{C_{control}}$ $T_{CTS} = \frac{I_{CTS}}{C_{control}}$

where $l_{ACK}, l_{RTS} and l_{CTS}$ is the length of ACK, RTS and CTS control packets respectively, MAC_{hdr} is the MAC header and PHY_{hdr} is the physical header. Authors in [6] proved the superiority of RTS/CTS in highly loaded network. His work is based on this 2-D Markov chain model. Most of the dynamic approaches for adaptive RTSThreshold are based on 2-D Markov model.

2.5.1 Related Works

The authors in[7], have evaluated the dependency of the RTS/CTS scheme on network size , however , without providing any general expression for the RTS/CTS threshold. But works in [8] and [9] has pointed out that the RTS/CTS handshake does not work as well as expected in theory. Approaches to fix the value of RT can be clustered main lying two types; Dynamic and Static. Authors in [10] has performed analysis to determine RT values for maximum performance and proposed static value [RT = 0] for all nodes, considering only single hop environment. On the other hand, dynamic approaches are discussed in [11], [12], [10], [13], [5], [14]. In Others, such as in [12] and [13] packet delivery ratio or transmission probability is emphasized. The common practice in [11], [12], [5], [14] is not to consider hidden node problem.

Chapter 3

Tuning RTS-Threshold

3.1 Related Terms Used with RTS-Threshold

3.1.1 Hidden Terminal Problem

Hidden nodes in a wireless network refer to nodes that are out of range of other nodes or a collection of nodes. Take a physical star topology with an access point with many nodes surrounding it in a circular fashion: Each node is within communication range of the AP, but the nodes cannot communicate with each other, as they do not have a physical connection to each other. In a wireless network, it is likely that the node at the far edge of the access point's range, which is known as A, can see the access point, but it is unlikely that the same node can see a node on the opposite end of the access point's range, B. These nodes are known as hidden. The problem is when nodes A and B start to send packets simultaneously to the access point. Since node A and B can not sense the carrier, Carrier sense multiple access with collision avoidance (CSMA/CA) does not work, and collisions occur, scrambling data. To overcome this problem, handshaking is implemented in conjunction with the CSMA/CA scheme.

In the basic transmission scheme due to the fact that carrier sensing range is almost equal to transmission range of a node, it effectively increases the probability of collisions. The problem of a station not being able to detect a potential competitor for the medium because the competitor is too far away(based on their carrier sensing range) is called the hidden node problem.

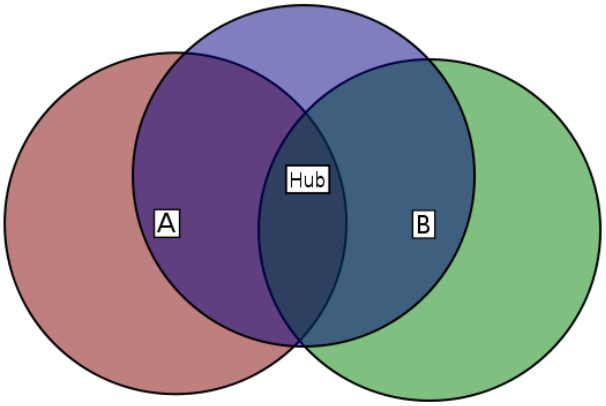

FIGURE 3.1: Hidden Terminal Problem

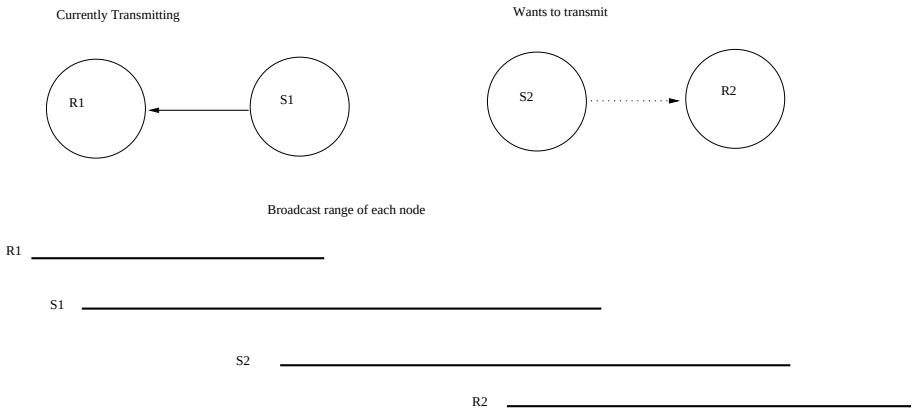

FIGURE 3.2: Exposed Terminal Problem

3.1.2 Exposed Teminal problem

In wireless networks, the exposed node problem occurs when a node is prevented from sending packets to other nodes due to a neighboring transmitter. Consider an example of 4 nodes labeled R1, S1, S2, and R2, where the two receivers are out of range of each other, yet the two transmitters in the middle are in range of each other.

Here, if a transmission between S1 and R1 is taking place, node S2 is prevented from transmitting to R2 as it concludes after carrier sense that it will interfere with the transmission by its neighbor S1. However note that R2 could still receive the transmission of S2 without interference because it is out of range from S1.

3.2 Advantages of RTS-CTS mechanism

To alleviate hidden node problem, Karn [15] proposed two way handshaking protocol known as the RTS/CTS handshaking mechanism. Bharghaven et al. [16] proposed an improved protocol which is known as RTS-CTS-DATA-ACK handshaking mechanism. The advantages of this mechanism are

- To reduce frame collisions introduced by the hidden terminal problem
- Originally the protocol fixed the exposed terminal problem as well, but modern RTS/CTS includes ACKs and does not solve the exposed terminal problem. This mechanism helps to solve this problem only if the nodes are synchronized. When a node hears an RTS from a neighboring node, but not the corresponding CTS, that node can deduce that it is an exposed node and is permitted to transmit to other neighboring nodes

3.3 Disadvantages of RTS-CTS mechanism

Though the RTS/CTS mechanism is able to solve the hidden node problem and reduce packet collision probability, it has several disadvantages. Authors in [17] had discussed several of these disadvantages like Inhibiting non-interfering parallel transmission, False Blocking & and Virtual Jamming.

Inhibiting Non-interfering Parallel Transmission

RTS/CTS mechanism blocks some non-interfering transmission which would be possible with the basic mechanism without collision. Thus, it will cause the reduction of overall throughput.

False Blocking

In the false blocking problem a node can remain blocked during the whole interval of a non-existing conversation. The worse fact is that, it can trigger a chain of nodes to be blocked by the non-existing conversation.

Virtual Jamming

In the virtual jamming problem, a potential malicious node , can make use of the false blocking problem by sending short RTS packets in short periods virtually jamming the whole or a significant part of the network using relatively small power.

From the above discussion, we can realize that the RTS/CTS-based reservation scheme trades some problems (like the hidden node problem) for others (inhibition of parallel transmissions and exposure to virtual jamming attacks). While elimination of the interference caused by hidden nodes does have a positive impact on the network performance, the problems introduced by the RTS/CTS mechanism will tend to counterbalance those benefits. Therefore we need to continue with the both schemes (i.e. the basic scheme and the RTS/CTS based scheme) in a balanced way to reduce the probability of collision and at the same time to avoid the problems of RTS/CTS mechanism. This balance can be achieved if we can avoid using the RTS/CTS mechanism for a certain $\eta \times 100$ percent of the packets and use RTS/CTS mechanism for the rest $(1 - \eta) \times 100$ percent. η can be tuned to achieve best performance. Moreover the smaller $\eta percent$ packets should be transmitted using basic schemes without the RTS/CTS protection because the collision probability is less for the small sized packets.

3.4 Disadvantages of Using Fixed Value of RTS-Threshold

When the payload is large, the probability of collision is high, so it is beneficial to use RTS/CTS conversation. On the other hand, if the payload is small, the probability of collision is comparatively low and it is better to go with the basic scheme. Traditionally RTS- Threshold is set to a *fixed* small value. But setting to a fixed small value is not optimal for all network situations and it can not effectively inter-mix the two schemes over all the packets flowing through the network. The problem of having fixed RTS-Threshold can be described as follows. As the packet size of a network is random and not known before, with a fixed RTS-Threshold it may be happen that all the packets in the network are having sizes larger than that fixed RTS-Threshold value. Consequently, all the packets will be transmitted using RTS/CTS mechanism. Also the other way around may happen, for example all the packets may have sizes smaller than the fixed value of RTS-Threshold, causing all of them to be transmitted using the basic scheme. In both the cases we can not use the η percent rule, therefore can not intelligently inter-mix both schemes.

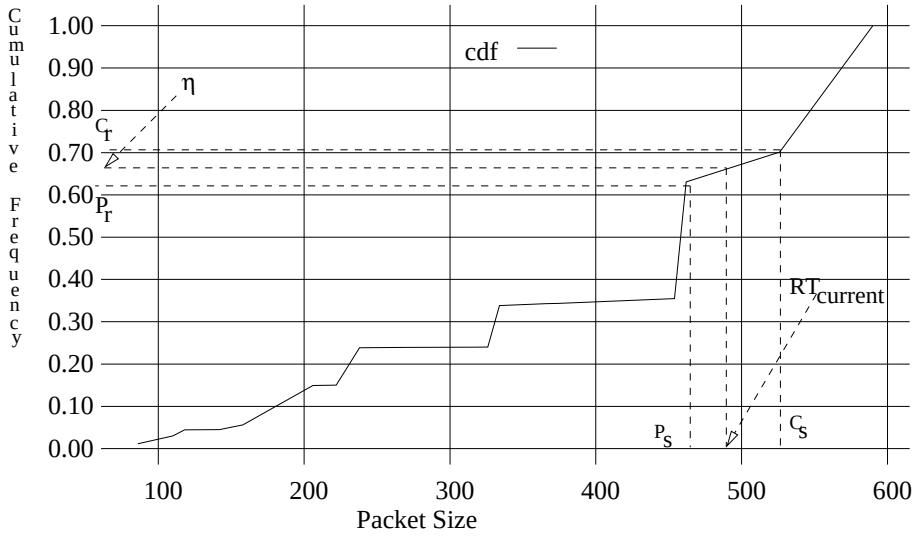

FIGURE 3.3: $RTS_{current}$ Calculation Using CDF

3.5 The Relative Definition of Small

We can adaptively set the value of RTS-Threshold based on network traffic then intermixing these two schemes can be easily achieved. Our main proposal is too use basic scheme for *relatively* small sized packets and use RTS/CTS mechanism for *relatively* large size packets. To incorporate this idea, the value of RTS-Threshold needs to be intelligently set to a value such that $\eta \times 100$ percent of packet's size fall below that value. Mathematically it can be described as follows: suppose the sizes of packets flowing through a node are $s_1, s_2, s_3, \ldots, s_n$ (in ascending sorted order) with probability $p_1, p_2, p_3, \ldots, p_n$. Then the value of RTS-Threshold is set to a value such that:

$$Pr\{S \leq RTS - Threshold\} = \eta$$

where S is a random variable denoting packet size.

A node at first learns the sizes of the packets it is generating or forwarding as an intermediate node for a certain time interval. Then it sets the value of RTS-Threshold for the next interval using the above equation. It also continues it's learning process in the subsequent intervals and adjusts the RTS-Threshold dynamically from one interval to another. The details of the algorithm is as follows.

3.6 The Algorithm

To set the RTS-Threshold dynamically, we need to observe the current distribution of packet size in the network. But for this, we need information from all the active nodes. This necessiates inter-layer communication(Routing-MAC layer). To avoid this overhead, we considered packet distribution sensed by individual nodes. By considering flow of packets through itself, a node evaluates the packet distribution in the neighbouring environment.

We equip each node with a *traffic observer* which runs in the background. Having received a packet p with size s_i, the node increments the frequency count f_i for the packet size s_i. Every δ seconds the traffic observer wakes up and calculates a new value of RTS-Threshold based on the packet distribution statistics collected within the last δ interval. To calculate new RT value, the observer at first rearranges the frequency count of packet sizes in increasing order of packet size. Suppose the total number of different packet sizes is n and S is a random variable denoting packet size. Let us denote P_i be the probability that a packet's size is less than or equal to s_i. Then, mathematically:

$$P_i = Pr\{S \leq s_i\} = \left(\frac{\sum_{j=1}^{i} f_j}{\sum_{k=1}^{n} f_k} \right)$$

Note that using the above equation $P_n = 1$. Actually P_i is the *cumulative distribution function* (CDF) for the different packet sizes which is depicted in Fig. 3.3.

Using this CDF, calculation of new RTS-Threshold is pretty simple. Let, P_r is the greatest probability less then η, P_s is the packet size at P_r, C_r is the least probability greater then η, and C_s is the packet size at C_r. Using linear interpolation the traffic observer calculates the current RTS-Threshold using the equation below (see Fig. 3.3):

$$RT_{current} = \left\lfloor P_s + \frac{(\eta - P_r) * (C_s - P_s)}{(C_r - P_r)} \right\rfloor$$

The average RTS-Threshold is updated as

$$RT_{average} = \lfloor \alpha * RT_{prev} + (1 - \alpha) * RT_{current} \rfloor$$

where, RT_{prev}=previous RTS-Threshold and α controls the relative weight of recent and past history of RTS-Threshold calculation. The value of α lies between 0 to 1.

Chapter 4

Validation by Simulation

4.1 Network Simulator 2

Network simulation software enable us to predict behavior of a large-scale and complex network system such as Internet at low cost under different configurations of interest and over long period. Many network simulators, such as NS2, Openet, Qualnet are widely available. We have used NS2 for this thesis. NS2 is a discrete event simulator written in C++, with an OTcl interpreter shell as the user interface that allows the input model files (Tcl scripts) to be executed. Most network elements in NS2 simulator are developed as classes, in object-oriented fashion. The simulator supports a class hierarchy in C++, and a very similar class hierarchy in OTcl. The root of this class hierarchy is the TclObject in OTcl. Users create new simulator objects through the OTcl interpreter, and then these objects are mirrored by corresponding objects in the class hierarchy in C++. NS2 provides substantial support for simulation of TCP, routing algorithms, queueing algorithms, and multicast protocols over wired and wireless (local and satellite) networks, etc. It is freely distributed, and all source code is available.

Developing new networking protocols and creating simulation scripts are complex tasks, which requires understanding of the NS2 class hierarchy, C++, and Tcl programming. However, in this thesis, we have designed and ran simulations in Tcl scripts using the simulator objects without changing NS2 core components such as class hierarchy, event schedulers, and other network building blocks. The only change has been in the codes of IEEE 802.11 mac layer and it's associated timers.

4.2 Experimental Setup

We used an area of 1000×1000 square unit. We have done 2 different types of experiment . One of them runs for 1000 time units and another runs for 800 time unit. For each experiment we created 5 network topologies and have taken the average of the 5 results. All the experiments were done using CBR traffic sources. The environment used was multi-hop with the presence of hidden node. We considered a static network. For traffic generation we have used an uniform packet size generator with min and max size specified for each experiment. We did experiments in three network conditions; lite (50 nodes), medium (75 nodes), dense (100 nodes). For performance measurement we have calculated the overall (aggregate) throughput of all the nodes.

4.3 Performance Metrics

We have chosen throughput as our performance matrix. Throughput is calculated as follows:
Throughput=sum of all data packets successfully received by the destination nodes/total simulation time;
We compared the throughput evaluated at different $\eta(Ratio)$. This $\eta(Ratio)$ defines throughput under basic scheme, throughput where all packets are transmitted maintaining RTS/CTS scheme and throughput where different percent of packets gets the benefit of RTS/CTS mechanism.

4.4 Scenario Generation

In order to carry out meaningful study of different networking issues like protocol interaction, congestion control, effect of network dynamics, scalability etc it is necessary to carry simulations on the right kind of scenario. Different scenarios can be used to illustrate/compare interesting network performances. The NS scenario generator can be used to create different random scenarios for simulation. We have taken area of 1000*1000. Our scenarios are consist of 50 nodes,75 nodes and 100 nodes and most importantly we have considered static network. The command in NS-2 is Setdest to generate the scenario.

$./setdest - n < num\_of\_nodes > -p < pausetime > -s < maxspeed > -t < simtime > -x < maxx > -y < maxy >>< outdir > / < scenario - file >$
With ,

maxx=1000
Maxy=1000
Num_of_nodes=50 or 75 or 100 depending on experiment
Simulation time =800 or 1000ns
Pausetime=1040ns (Actually we take a paustime greater than simulation time to get a static scenario)
Setdest command will generate a 1000*1000 topology with n nodes random distributed in static environment.

4.5 Traffic Generation

Random traffic connections of CBR can be setup between mobile nodes using a traffic-scenario generator script. In order to create a traffic-connection file, we need to define the type of traffic connection (CBR or TCP), the number of nodes and maximum number of connections to be setup between them, a random seed and in case of CBR connections, a rate whose inverse value is used to compute the interval time between the CBR pkts. So the command line looks like the following:

$nscbrgen.tcl[-typecbr|tcp][-nnnodes][-seedseed][-mcconnections][-raterate][-mtmax\_time] > output.tcl$

with,
type=cbr
nodes=n
seed=1
mc connection=200
rate =x
mt = z
cbrget.tcl creates traffic having rate x with n nodes. We also considered packets with different size from different nodes. The number of nodes are 50 or 75 or 100 depending on the experiment. Rate is 2.0 or 4.0 or 6.0. The mt value sets the highest time by which a source must start. It is normally set to 50 but in one experiment it is set to varying amount like 200, 400, 600, and 800.

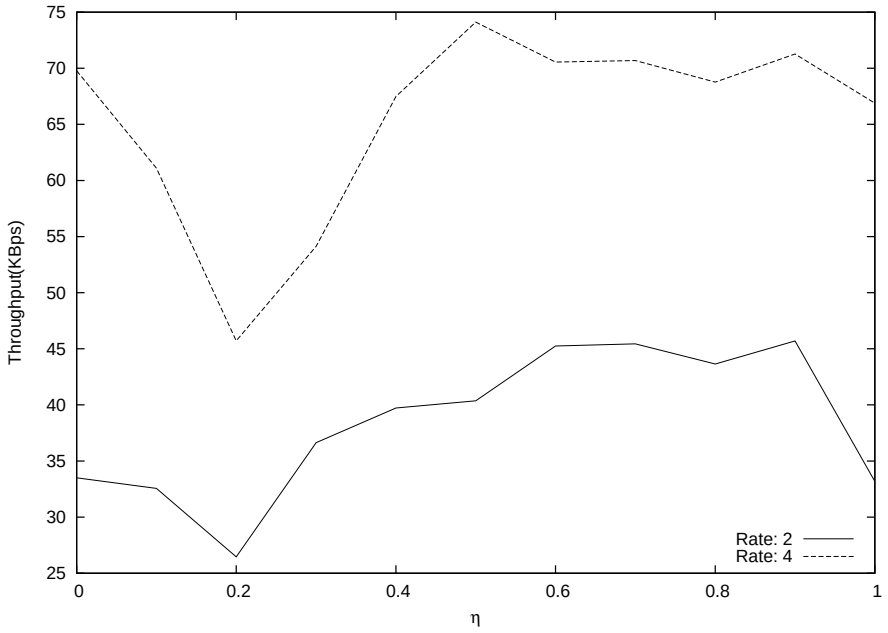

FIGURE 4.1: η vs Throughput Node: 50 Packet size: 8-512 Byte

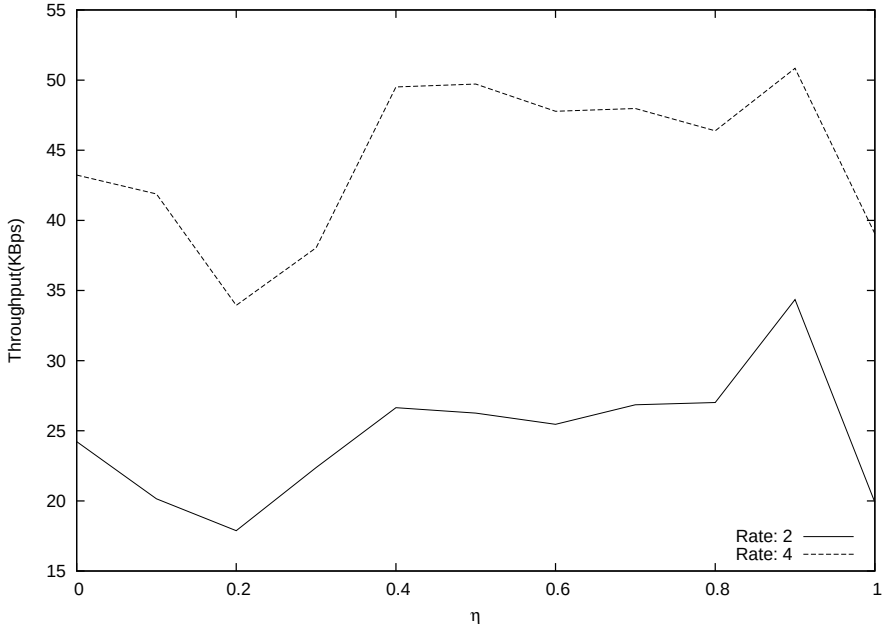

FIGURE 4.2: η vs Throughput Node: 50 Packet size: 16-512 Byte

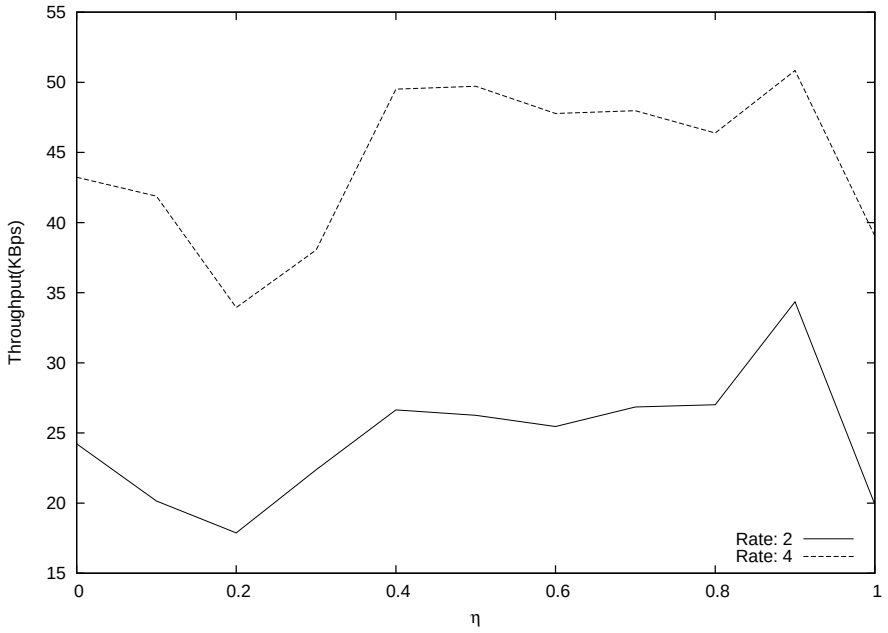

FIGURE 4.3: η vs Throughput Node: 50 Packet size: 32-512 Byte

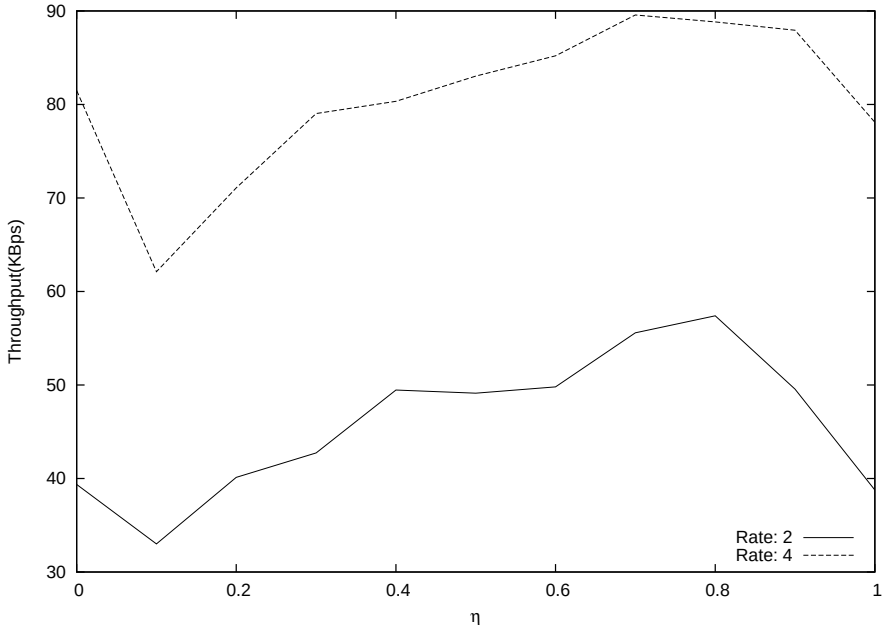

FIGURE 4.4: η vs Throughput Node: 50 Packet size: 64-512 Byte

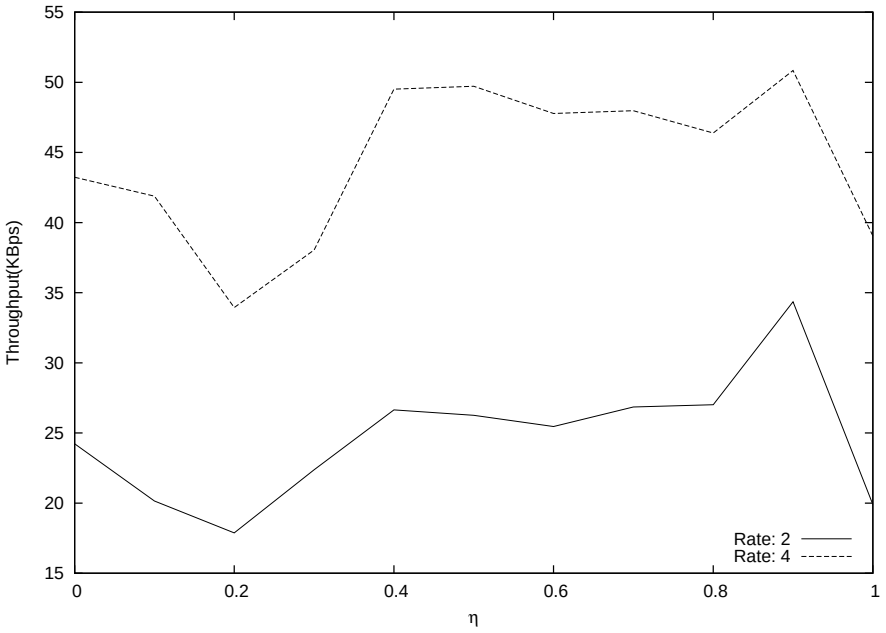

FIGURE 4.5: η vs Throughput Node: 75 Packet size: 8-512 Byte

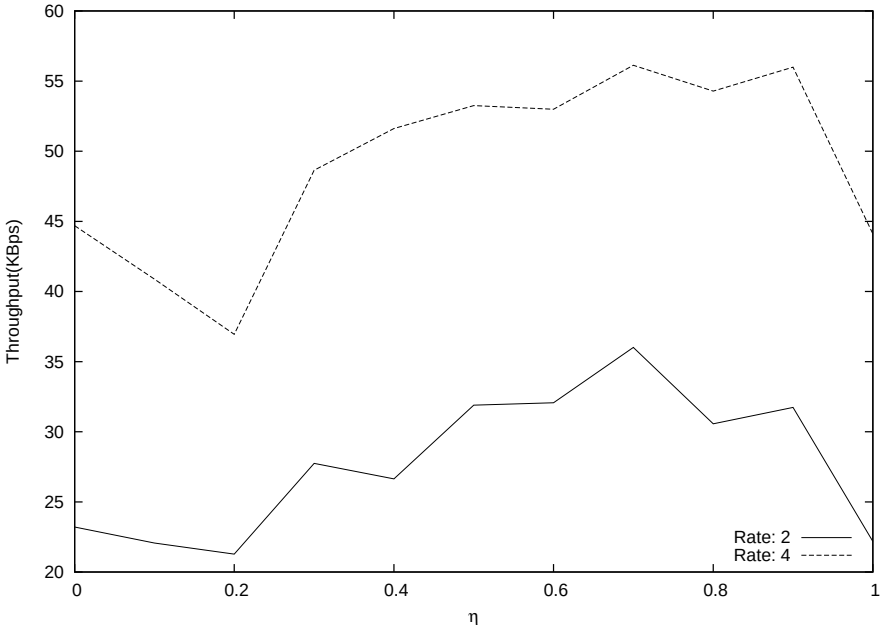

FIGURE 4.6: η vs Throughput Node: 75 Packet size: 16-512 Byte

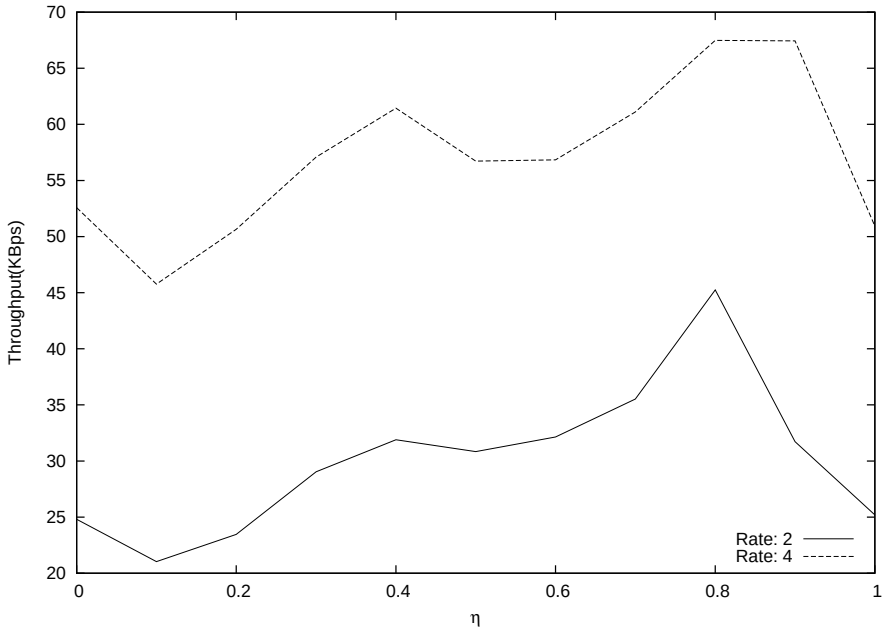

FIGURE 4.7: η vs Throughput Node: 75 Packet size: 32-512 Byte

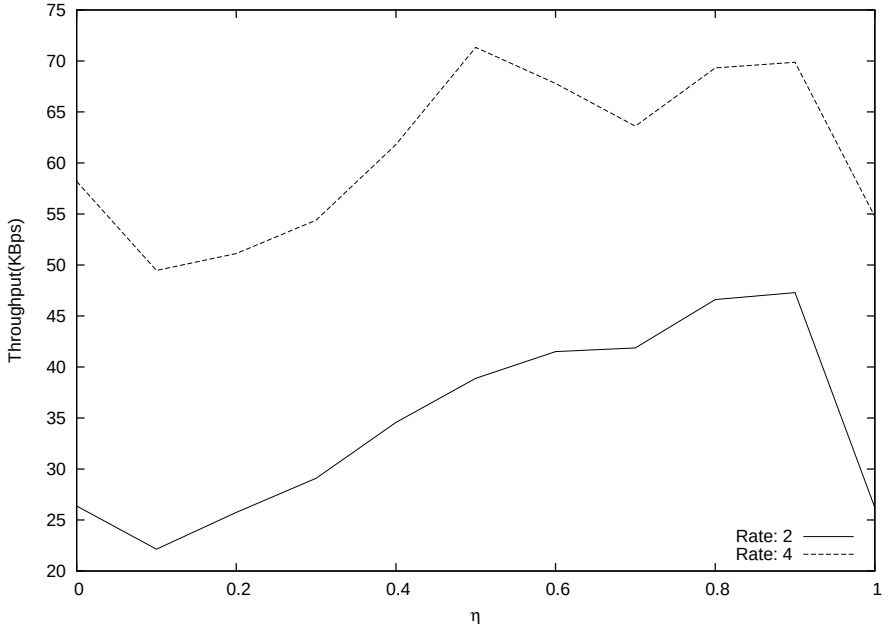

FIGURE 4.8: η vs Throughput Node: 75 Packet size: 64-512 Byte

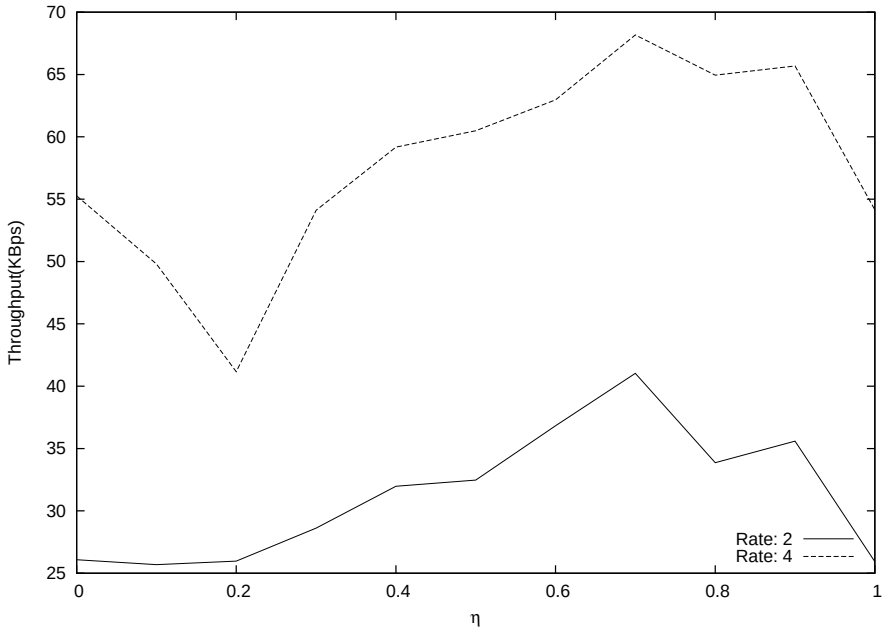

FIGURE 4.9: η vs Throughput Node: 100 Packet size: 8-512 Byte

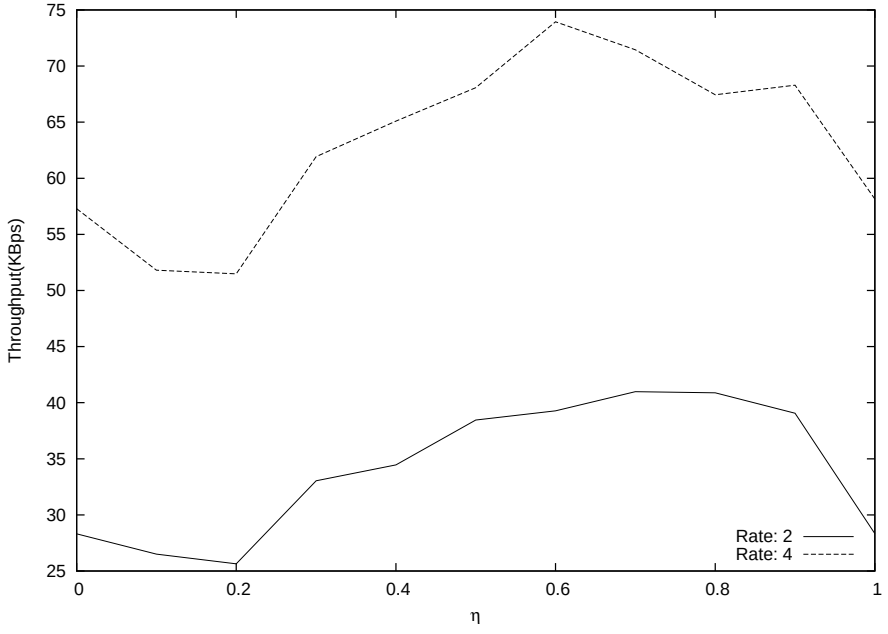

FIGURE 4.10: η vs Throughput Node: 100 Packet size: 16-512 Byte

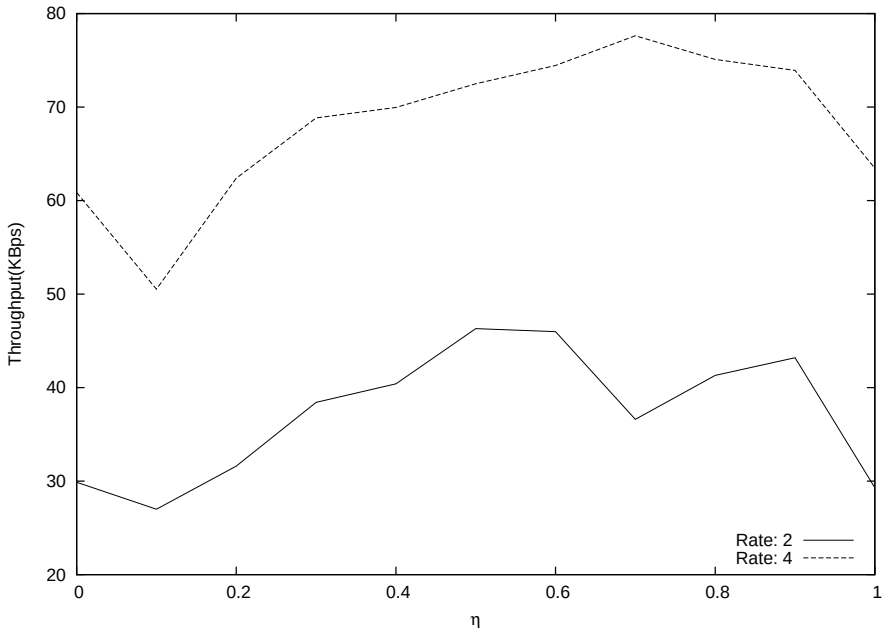

FIGURE 4.11: η vs Throughput Node: 100 Packet size: 32-512 Byte

4.6 Experimental Results

In static network , if nodes have similar distribution of payload, RTS-Threshold will converge to a fixed value eventually from its default value 0. Moreover if payload is distributed within a range, it is also reflected in RTS-Threshold value.

In Fig. 4.1, 4.2, and 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 4.10, and 4.11 overall throughput for different η values are shown. When the value of η is set to 0 then all packets are transmitted with RTS/CTS conversation. $\eta = 1$ indicates that all packets are transmitted with basic scheme without any RTS/CTS. The current IEEE 802.11 standard transmits all packets with RTS/CTS dialogue. So the value of the curve with $\eta = 0$, actually represents the performance of the current standard. It can be observed that the throughput is high for $0 \leq \eta \leq 1$ and in most of the experiments the optimum value is found for $0.6 \leq \eta \leq 0.8$.

In the graph of Fig. 4.12, 4.13, 4.14, 4.15, 4.16, 4.17, 4.18, 4.19, and 4.20 we see the effect of change in throughput for the change in packet size distribution. Here we have carefully devised the CBR generators to create 5 non overlapping time intervals of 200 time unit each. In each of the intervals, nodes generate packets of different sizes. In first interval, packet size is uniformly distributed between $8 - 512$ byte. In second interval it is from $16 - 512$ byte and so on. We have turned off all the generators at the end of each interval and a new set of generators are started uniformly in the next interval. This has created a *sea-saw* effect on the performance curve. There is a sharp

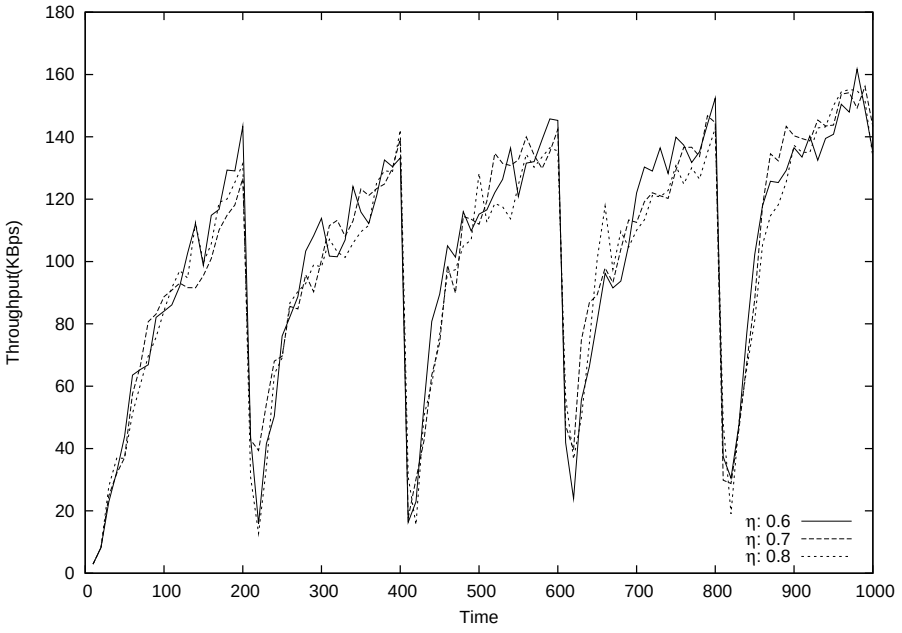

FIGURE 4.12: Instantaneous Throughput Node: 50 Window Size δ= 10

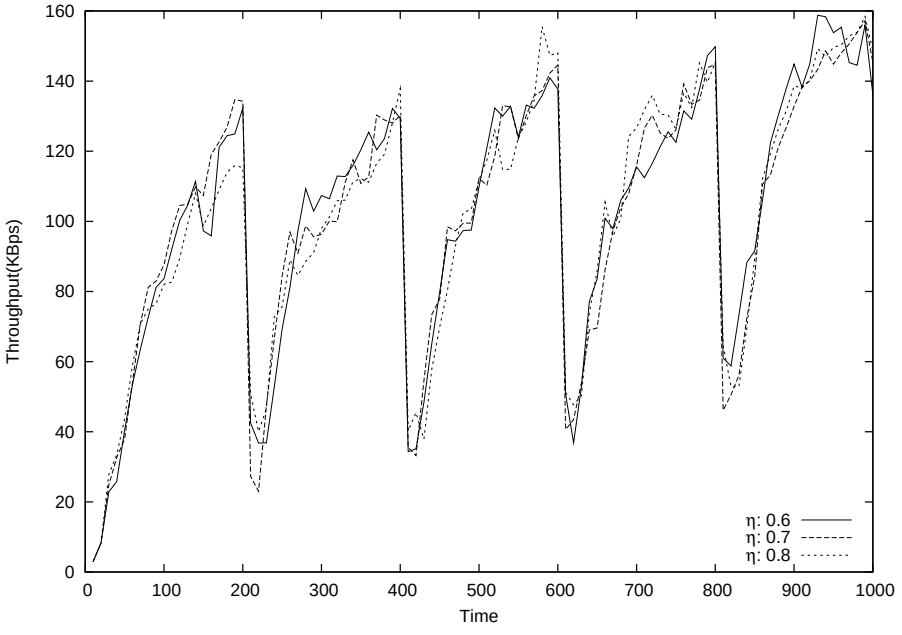

FIGURE 4.13: Instantaneous Throughput Node: 50 Window Size δ= 20

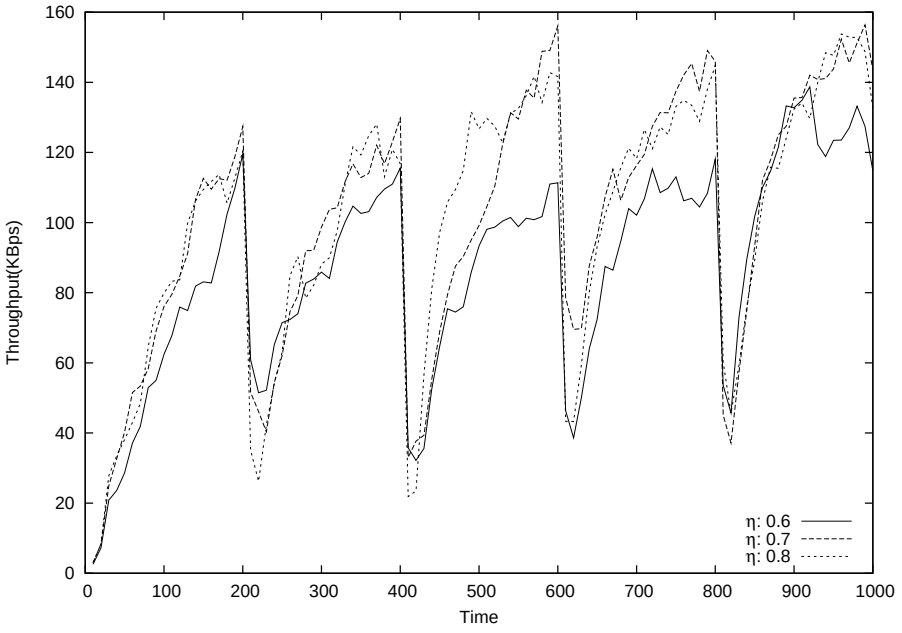

FIGURE 4.14: Instantaneous Throughput Node: 50 Window Size δ= 50

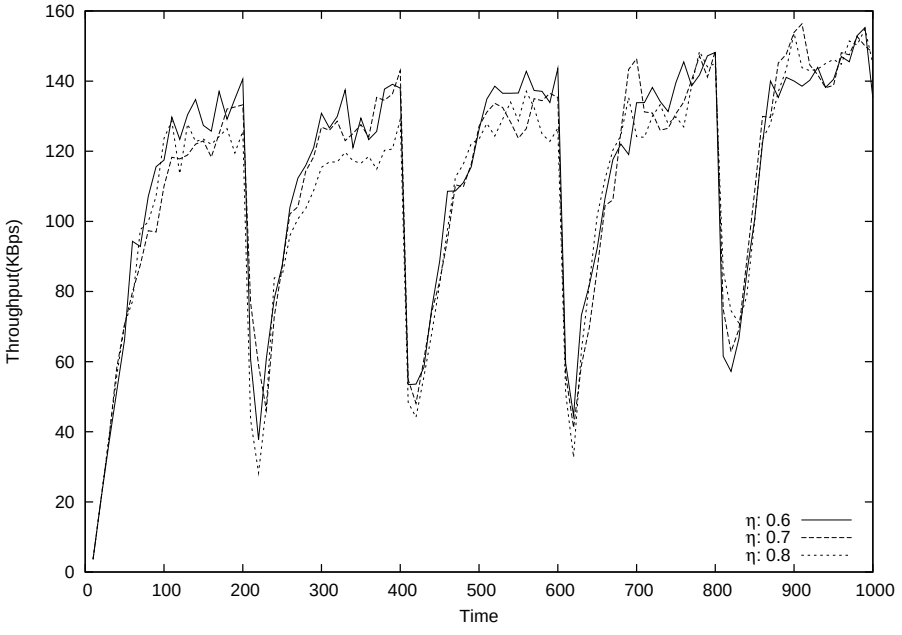

FIGURE 4.15: Instantaneous Throughput Node: 75 Window Size δ= 10

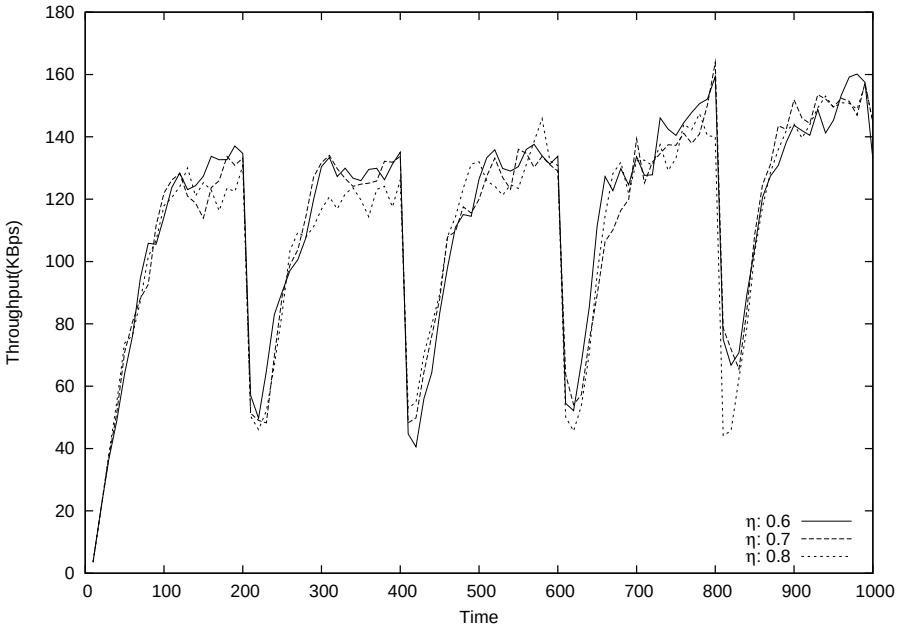

FIGURE 4.16: Instantaneous Throughput Node: 75 Window Size δ= 20

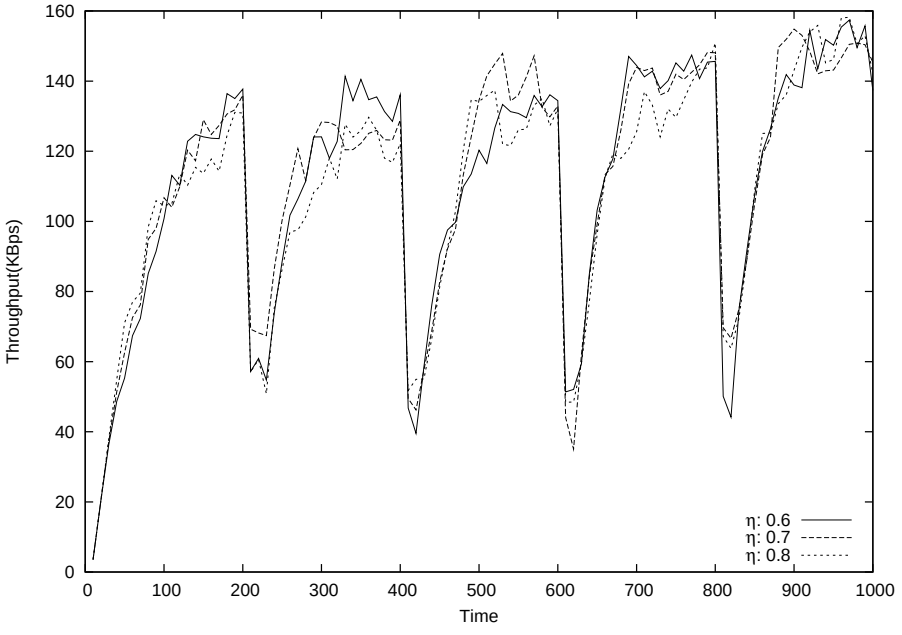

FIGURE 4.17: Instantaneous Throughput Node: 75 Window Size δ= 50

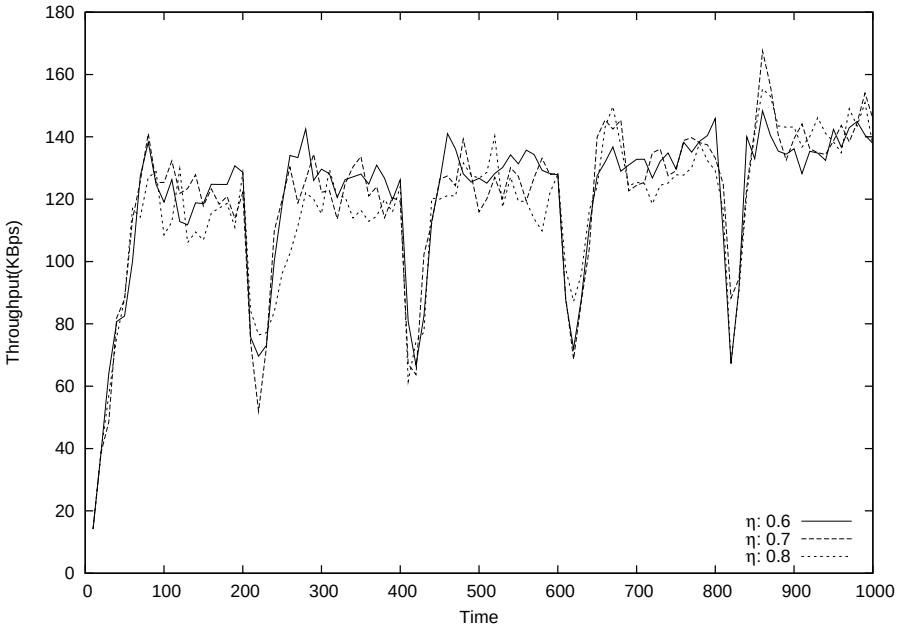

FIGURE 4.18: Instantaneous Throughput Node: 100 Window Size δ= 10

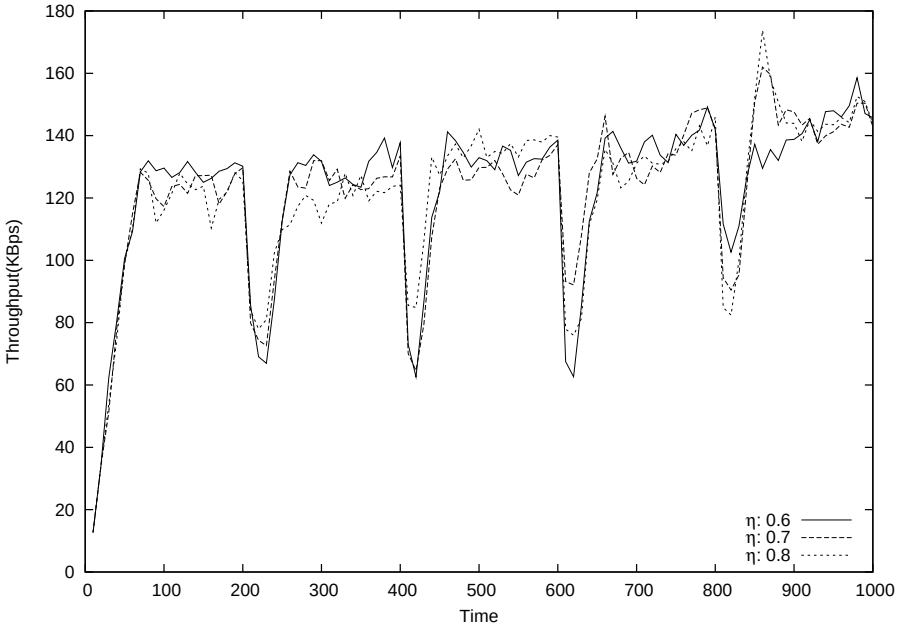

FIGURE 4.19: Instantaneous Throughput Node: 100 Window Size δ= 20

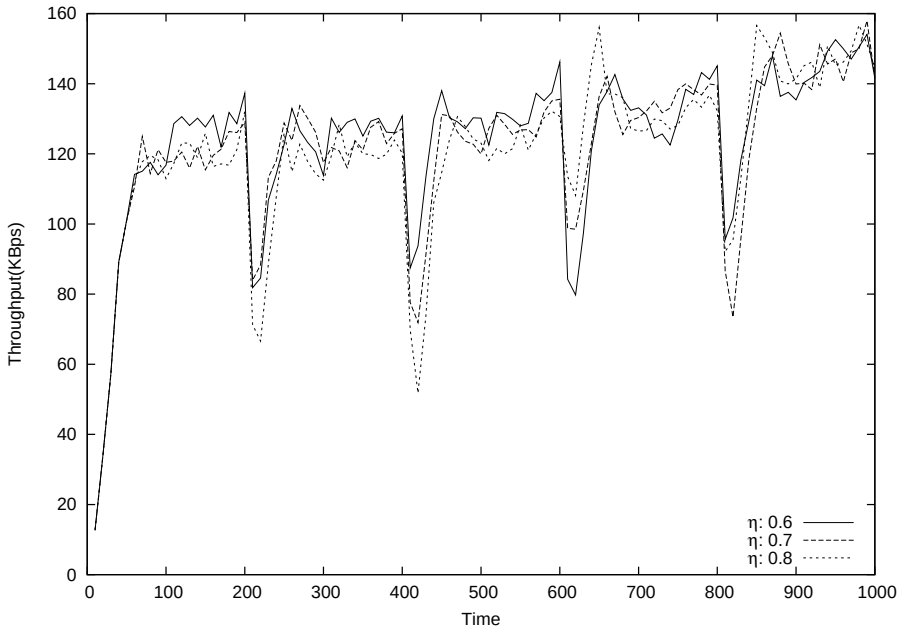

FIGURE 4.20: Instantaneous Throughput Node: 100 Window Size δ= 50

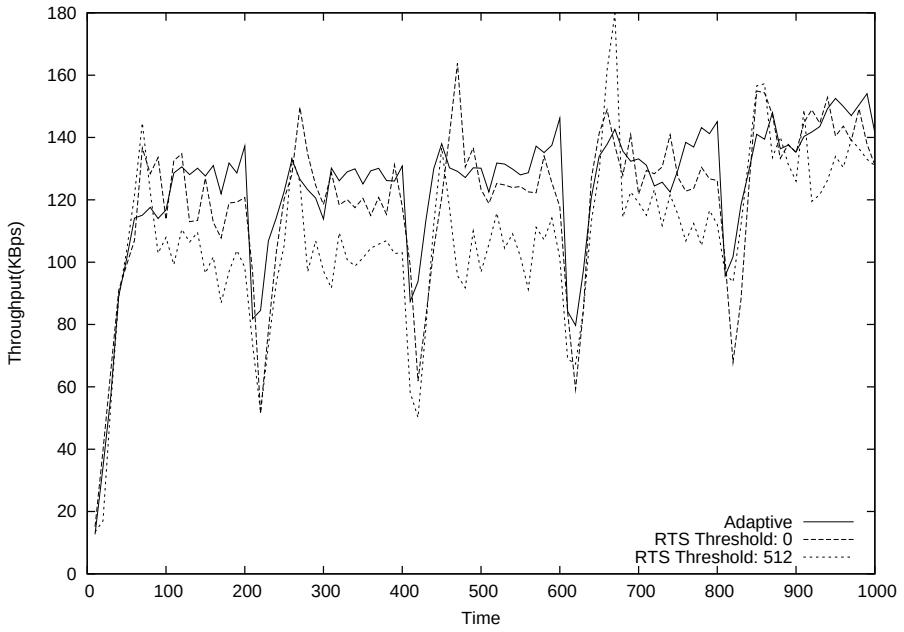

FIGURE 4.21: Comparison between Adaptive & Fixed RTS Threshold Scheme

fall in throughput at the end of each interval which rises again in the next new interval as more and more traffic sources are started.

In Fig. 4.21 our proposed adaptive scheme is compared with the scheme that uses *fixed* RTS Threshold value of 0 and 512 in an environment with 100 nodes and different packet

distributions. It is easy to see that our proposed adaptive scheme outperforms all such *non-adaptive* schemes.

Chapter 5

Conclusion and Future Works

In this research we propose an adaptive scheme to effectively use RTS/CTS handshaking in IEEE 802.11. We proposed a dynamic way to adjust the RTS Threshold based on current packet distribution of the network. Through simulation we validate our proposal. Evaluated results in NS-2 showed that the proposed adaptive scheme achieves better result than the current IEEE 802.11 scheme. We further intend to experiment on mobile networks.

Appendix A

NS2 The Network Simulator

A.1 What is NS2

NS is an object-oriented,discrete event simulator targeted at networking research. NS provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks. Later NS-2 (version 2) was developed at UC Berkeley in C++ and OTcl (Object-oriented extension of Tcl)

A.2 Installing NS2

- Download NS2 version 2.31 from http://www.isi.edu/nsnam/ns/ns-build.html website
- Unzip the tar archive
- Open Terminal and Go to the ns2 root directory. Here /ns-allinone-2.31
- Run ./install script

A.3 Set Path

After installation to set the path type the following contents.

o Export LD_LIBRARY_PATH=~ns-allinone-2.31/otcl-1.13

o Export LD_LIBRARY_PATH=~ns-allinone-2.31/ns-2.31/lib

o Export TCL_LIBRARY=~ns-allinone-2.31/tcl8.4.14/library

To access NS command in any directory

o Go to ns2.31 directory under the ns-allinone-2.31
o Execute the command make install

To access setdest command in any directory

o Go to ns2.31/indep-utils/cmu-scen-gen/setdest directory under the ns-allinone-2.31
o Execute the command make install

A.4 Validation

* Open terminal in ˜ns-allinone-2.31/ns-2.31
* Run ./validate
* If installation and path variable setting are done properly, then the validation will be successful

A.5 Scenario Generation

* This is done using the setdest command. There are two versions for setdest. Version 2 is most recently implemented and used.
* Version 2 signature of setdest command is
setdest -v < 2 > -n < *nodes* > -s < *speedtype* > -m < *minspeed* > -M < *maxspeed* > -t < *simulationtime* > -P < *pausetype* > -p < *pausetime* > -x < *maxX* > -y < *maxY* >

o -v version number; Here 2

o -n number of nodes. Generated node number will be 0 to (n-1)

o -s speed type (uniform, normal); s=1 uniform speed from min to max; s=2 normal speed clipped from min to max

o -m minimum speed > 0

o -M maximum speed

o -P pause type (constant, uniform); P=1 constant pause; P=2 uniform pause [0, 2*p]

o -p pause time (a median if uniform is chosen)

o -x x dimension of space

o -y y dimension of space

* After running the command a scenario will be generated. Pipe the scenario in file

* Example

o setdest -v 2 -n 10 -m 10 -M 100 -t 20 -P 1 -p 10 -x 200 -y 400 > scen-exp1

* For Further Explanation: Go through the source code ˜ns-allinone-2.31/ns-2.31/indep-utils/cmu-scen-gen/setdest/setdest.cc

A.6 CBR Traffic Generation

* This is done with the help of cbrgen.tcl file executing with ns command

* Open terminal in ˜ns-allinone-2.31/ns-2.31/indep-utils/cmu-scen-gen

* ns cbrgen.tcl [-type cbr—tcp] [-nn nodes] [-seed seed] [-mc connections] [-rate rate]

o -type traffic type (tcp, udp/cbr)

o -nn the highest node number(node number will be 0 to nn)

o -seed seed for random variable generation which is used to create random number of source-destination pair

o -mc maximum number of connections; i.e.; source-destination pair

o -rate it is the inverse of the interval between packet transmission & should be <0

* After running the command a cbr traffic will be generated. Pipe the traffic in file

* Example

o ns cbrgen.tcl -type cbr -nn 9 -seed 1 -mc 10 -rate .25 > cbr-exp1

* For Further Explanation: Go through the source code ˜ns-allinone-2.31/ns-2.31/indep-utils/cmu-scen-gen/cbrgen.tcl

A.7 Writing Simulation Generation File

* Download this sample tcl file or write a new tcl file.This sample should give some idea of the different parameters that need to be configured for a typical simulation. A wireless simulation is made up of some number of MobileNodes. Each such mobile node needs some options to be configured like routing protocol, MAC layer protocol, antenna type, channel type etc. A list of all parameters is given below.

o Channel type (Channel/WirelessChannel) ;# Set channel for wireless media

o Propagation model (Propagation/TwoRayGround);

o Interface type (Phy/WirelessPhy) ;

o MAC layer protocol (Mac/802_11) ;# The name of MAC layer protocol

o Routing protocol (AODV) ;# The name of routing protocol

o Interface Queue type (CMUPriQueue - for DSR) ;

- o Interface Queue Length (50) ;# Queue length of a node for storing packets
- o Antenna type (Antenna/OmniAntenna) ;
- o LL type (LL) ;# Link Layer type

* Set the required parameters in the tcl file for the specific scenario and traffic. These are the configuration parameters for the topology structure, like the dimensions of the grid, number of nodes present etc. A list of all of them is given below for reference.
Example(For the previous scenario and traffic)

- o set val(x) 200 ;# X dimension of the topography
- o set val(y) 400 ;# Y dimension of the topography
- o set val(ifqlen) 50 ;# max packet in ifq
- o set val(seed) 0.0 ;
- o set val(adhocRouting) AODV ;# The routing protocol
- o set val(nn) 10 ;# how many nodes are simulated
- o set val(cp) "cbr-exp1" ;# the absolute or relative path of the traffic file
- o set val(sc) "scen-exp1" ;# the absolute or relative path of the scenario file
- o set val(stop) 200.0 ;# the simulation time

* Turn on the trace for the required layers
Example

- o -agentTrace ON {}
- o -routerTrace ON {}
- o -macTrace ON {}
- o -movementTrace OFF{}

* For generating the trace file in new format, remove the comment or vice versa
$ns_ use-newtrace
* Set the output trace file with absolute or relative path
Example: set tracefd [open exp1.tr w]

A.8 Trace Generation

* Paste the tcl file to the ~ns-allinone-2.31/ns-2.31/tcl/ex directory

* Open terminal in ~ns-allinone-2.31/ns-2.31/tcl/ex

* Execute commandns < *tclscript* >

Example: ns sample.tcl

* If the simulation is successful, the desired trace file will be generated.

A.9 Trace Analysis

Explanation of new trace format

The new trace format as seen above can be can be divided into the following fields :

Event type: In the traces above, the first field (as in the older trace format) describes the type of event taking place at the node and can be one of the four types:
s send
r receive
d drop
f forward

General tag: The second field starting with "-t" may stand for time or global setting
-t time
-t * (global setting)

Node property tags: This field denotes the node properties like node-id, the level at which tracing is being done like agent, router or MAC. The tags start with a leading "-N" and are listed as below:
-Ni: node id
-Nx: nodes x-coordinate
-Ny: nodes y-coordinate
-Nz: nodes z-coordinate
-Ne: node energy level
-Nl: trace level, such as AGT, RTR, MAC
-Nw: reason for the event. The different reasons for dropping a packet are given below:
"END" DROP_END_OF_SIMULATION
"COL" DROP_MAC_COLLISION
"DUP" DROP_MAC_DUPLICATE
"ERR" DROP_MAC_PACKET_ERROR
"RET" DROP_MAC_RETRY_COUNT_EXCEEDED
"STA" DROP_MAC_INVALID_STATE
"BSY" DROP_MAC_BUSY

"NRTE" DROP_RTR_NO_ROUTE i.e no route is available.

"LOOP" DROP_RTR_ROUTE_LOOP i.e there is a routing loop

"TTL" DROP_RTR_TTL i.e TTL has reached zero.

"TOUT" DROP_RTR_QTIMEOUT i.e packet has expired.

"CBK" DROP_RTR_MAC_CALLBACK

"IFQ" DROP_IFQ_QFULL i.e no buffer space in IFQ

"ARP" DROP_IFQ_ARP_FULL i.e dropped by ARP

"OUT" DROP_OUTSIDE_SUBNET i.e dropped by base stations on receiving routing updates from nodes out-side its domain.

Packet information at IP level: The tags for this field start with a leading "-I" and are listed along with their explanations as following:
-Is: source address.source port number
-Id: dest address.dest port number
-It: packet type
-Il: packet size
-If: flow id
-Ii: unique id
-Iv: ttl value

Next hop info: This field provides next hop info and the tag starts with a leading "-H".
-Hs: id for this node
-Hd: id for next hop towards the destination.

Packet info at MAC level: This field gives MAC layer information and starts with a leading "-M" as shown below:
-Ma: duration
-Md: dsts ethernet address
-Ms: srcs ethernet address
-Mt: ethernet type

Packet info at "Application level": The packet information at application level consists of the type of application like ARP,TCP, the type of adhoc routing protocol like DSDV, DSR, AODV etc being traced. This field consists of a leading "-P" and list of tags for different application is listed as below:
-P arp Address Resolution Protocol. Details for ARP is given by the following tags:

-Po: ARP Request/Reply
-Pm: src mac address
-Ps: src address
-Pa: dst mac address
-Pd: dst address
-P dsr This denotes the adhoc routing protocol called Dynamic source routing. Information on DSR is represented by
the following tags:
-Pn: how many nodes traversed
-Pq: routing request flag
-Pi: route request sequence number
-Pp: routing reply flag
-Pl: reply length
-Pe: src of srcrouting− >dst of the source routing
-Pw: error report flag
-Pm: number of errors
-Pc: report to whom
-Pb: link error from linka− >linkb
-P cbr Constant bit rate. Information about the CBR application is represented by the following tags:
-Pi: sequence number
-Pf: how many times this pkt was forwarded
-Po: optimal number of forwards
-P tcp Information about TCP flow is given by the following subtags:
-Ps: seq number
-Pa: ack number
-Pf: how many times this pkt was forwarded
-Po: optimal number of forwards
This field is still under development and new tags shall be added for other applications as they get included along the way.

A.10 Links to related sites

- Home: http://www.isi.edu/nsnam/ns/index.html
- Tutorial: http://www.isi.edu/nsnam/ns/tutorial/nsindex.html
- Download: http://www.isi.edu/nsnam/ns/ns-build.html
- Installation: http://teacher.buet.ac.bd/ashikur/ns2/

- Simulation: http://teacher.buet.ac.bd/ashikur/ns2/#sim
- Gnuplot: file: http://t16web.lanl.gov/Kawano/gnuplot/index-e.html

Appendix B

Codes Modified

B.1 Mac Layer Modification

B.1.1 File mac-802_11.h

```
class MAC_MIB {
        .
        .
        .
        public:

                int             Adaptive;
                int             Window_Size;
                int             Window_Start_Time;
                double          Alpha;
                int             File_Enable;
                int             Array;
                int             Print;
                int             Cdf;
                double          Ratio;
        .
        .
        .
        public:
                inline void setRTSThreshold(int RTS){RTSThreshold=RTS;}

        };
```

```
class Mac802_11 : public Mac {
                .
                .
                .
```

```
        friend class WindowTimer;
        .
        .
        .
    protected:
        void    windowHandler(void);
        .
        .
        .
    private:
        void    window_timer(void);
        .
        .
        .

        int flag;
        int time;
        ofstream summeryfile;
        ofstream outputfile;
        int output[1000];
        map<int,int> op;
        long total_count;
        int totalreceivedbytes;
    };
```

B.1.2 File mac-802_11.cc

```
Mac802_11::Mac802_11() :
    Mac(), phymib_(this), macmib_(this), mhIF_(this), mhNav_(this),
    mhRecv_(this), mhSend_(this),
    mhDefer_(this), mhBackoff_(this), mhBeacon_(this), mhProbe_(this),
    mhWindow_(this)
{
    .
    .
    .
    for(int i=0; i< 1000 ; i++)
    {
        output[i]=0;
    }
    flag=1;
    totalreceivedbytes=0;


    char buff[5];
```

```
        sprintf(buff,"sum");
        summeryfile.open (buff,ios::app);
        summeryfile.close();

        sprintf(buff,"%d",index_);
        if(macmib_.File_Enable==1)
        {
        outputfile.open (buff,ios::app);
                outputfile.close();
        }

        if(macmib_.Adaptive != 0)
        {
                mhWindow_.start(macmib_.Window_Start_Time);
                if(macmib_.Print)printf("Adaptive Algorithm Started for:
                %d with window %d.\n",index_,macmib_.Window_Size);
        }

        if(macmib_.Cdf)total_count = 0;
        time = 10;
}


void
Mac802_11::windowHandler()
{
        window_timer();
}

void
Mac802_11::window_timer()
{
        static int old_max_size = 0;
        static int cdf = 0;

        if(!macmib_.Cdf)
        {

                int i;
                int max = -1;
                int max_size = -1;
                if(macmib_.Array){
                        for(i = 0; i < 1000; i++)
                        {
                                if(output[i] > max)
                                {
                                        max = output[i];
                                        max_size = i;
```

```
                        }
                }
        }else{
                map<int,int>::iterator x;
                for(x=op.begin();x!=op.end();x++){
                        if(x->second > max)
                        {
                                max = x->second;
                                max_size = x->first;
                        }
                }
        }
        if(macmib_.Adaptive==1)max_size+=4;
        else if(macmib_.Adaptive==2)max_size-=4;
        if(max_size>8)
        {
                if(old_max_size)old_max_size = int(
macmib_.Alpha*old_max_size + (1-macmib_.Alpha)*max_size);
                else
                {
                        old_max_size = max_size;
                }
                macmib_.setRTSThreshold(old_max_size);
                if(macmib_.Print)printf("RTSThreshold: %d for
                %d.\n",macmib_.getRTSThreshold(),index_);
        }
        if(macmib_.Array)for(i = 0; i < 1000; i++)output[i] = 0;
        else op.clear();

}
else if(macmib_.Cdf)
{
        float total_prob = 0.0;
        int i,prev=0,current=0,RTS=0;
        float prev_prob=0.0,current_prob=0.0;
        double check;
        if(macmib_.Adaptive == 1)check = macmib_.Ratio;
        else if(macmib_.Adaptive == 2)check = 1-macmib_.Ratio;

        if(macmib_.Array)
        {
                if(total_count>0)
                {
                        for(i=0;i<1000;i++)
                        {
                                if(output[i]!=0)
                                {
                                        current=i;
```

```
                                        total_prob +=
                                        (float)output[i]/
                                        (float)total_count;

                                        current_prob=total_prob;

                                        if(cdf)
                                        {
                                                printf("%d      %f\n",
                                                current,total_prob);
                                        }

                                        if(total_prob>=check) break;
                                        prev=current;
                                        prev_prob=current_prob;
                                }
                                if(total_prob>=check) break;
                        }

                        if(macmib_.Cdf == 1)RTS =int(prev+
                        (((check-prev_prob)*(current-prev))/
                        (current_prob-prev_prob)));
                        else if(macmib_.Cdf == 2)RTS = current;
                }
        }
        else{
                        if(total_count>0){
                        map<int,int>::iterator current,prev;
                        map<int,int>::iterator x;
                        for(x=op.begin();x!=op.end();x++)
                        {
                                current = x;
                                total_prob += (float)
                                x->second/(float)total_count;
                                current_prob=total_prob;
                                if(total_prob>=check)break;
                                prev = current;
                                prev_prob=current_prob;
                        }
                        RTS = int(prev->first +
                        (((check-prev_prob)*(current->first-prev->first))
                        /(current_prob-prev_prob) ) );
                }

        if(macmib_.Adaptive==1)RTS+=4;
        else if(macmib_.Adaptive==2)RTS-=4;
        if(RTS>8)
        {
```

```
                        if(old_max_size)old_max_size = int(
                macmib_.Alpha*old_max_size + (1-macmib_.Alpha)*RTS);
                        else
                        {
                                old_max_size = RTS;
                        }
                        macmib_.setRTSThreshold(old_max_size);
                        if(macmib_.Print)printf("RTSThreshold: %d for %d.\n",
                        macmib_.getRTSThreshold(),index_);
                }
                if(macmib_.Array)for(i = 0; i < 1000; i++)output[i] = 0;
                else op.clear();
                total_count=0;
        }

        mhWindow_.start(double(macmib_.Window_Size));
        cdf++;
}

void
Mac802_11::recv_timer()
{
        .
        .
        .
        if (netif_->node()->energy_model() &&
        netif_->node()->energy_model()->adaptivefidelity()) {
        src = ETHER_ADDR(mh->dh_ta);
        netif_->node()->energy_model()->add_neighbor(src);
        }
        if(strcmp(packet_info.name(ch->ptype()),"cbr")==0)
        {
                char buff[5];
                int size =ch->size();
                sprintf(buff,"%d",index_);
                if(macmib_.File_Enable==1)
                {
                        outputfile.open (buff,ios::app);
                        outputfile <<size<<"\n";
                        outputfile.close();
                }
                if(macmib_.Array) output[size]++;
                else{
                        if(op.find(size)==op.end()) op.insert(make_pair(size,1));
                        else op[size]++;
                }
                if(macmib_.Cdf)total_count++;
                .
```

```
                .
                .
        }
        .
        .
        .
}

void
Mac802_11::recvDATA(Packet *p)
{
        .
        .
        .
        if(strcmp(packet_info.name(ch->ptype()),"cbr")==0)
        {

                totalreceivedbytes+=size;
                double now = Scheduler::instance().clock();
                if(now>=time){
                        char buff[80];
                        sprintf(buff,"%d",time);

                        summeryfile.open (buff,ios::app);
                        summeryfile<<totalreceivedbytes<<"\n";

                        summeryfile.close();
                        time+=10;
                        totalreceivedbytes = 0;
                }
        }
        .
        .
        .
}
```

B.2 Linking TCL variables to C++

There comes time, when one has to set parameter values while running simulation in NS2. This can be easily done by linking a TCL variable to a C++ variable and set the value of TCL variable which will be later on the value of C++ variable. For instance we want to set the window size by a parameter. For this we will use a TCL variable Window_Size_ and will set its value shown below.

wireless.tcl

```
.
.
.
Mac/802_11 set RTSThreshold_ 1000
Mac/802_11 set Adaptive_ 1
Mac/802_11 set Cdf_ 1
Mac/802_11 set Ratio_ 0.6
Mac/802_11 set Window_Size_ 50
Mac/802_11 set Window_Start_Time_ 50
Mac/802_11 set Alpha_ .6
Mac/802_11 set File_Enable_ 0
Mac/802_11 set Array_ 1
Mac/802_11 set Print_ 0
.
.
.
```

Next comes linking this variable. Let we want to link this variable to a C++ variable Window_Size.

mac-802_11.c

```
MAC_MIB::MAC_MIB(Mac802_11 *parent)
{
        .
        .
        .
        parent->bind("Adaptive_", &Adaptive);
        parent->bind("Window_Size_", &Window_Size);
        parent->bind("Window_Start_Time_", &Window_Start_Time);
        parent->bind("Alpha_", &Alpha);
        parent->bind("File_Enable_", &File_Enable);
        parent->bind("Array_", &Array);
        parent->bind("Print_", &Print);
        parent->bind("Cdf_", &Cdf);
        parent->bind("Ratio_", &Ratio);
}
```

As we have worked with MAC layer we used the MAC_MIB function of mac802_11.cc file. Here the code to bind the Window_Size_ variable to Window_Size variable is shown. During simulation Window_Size will have value that is set to Window_Size_.

B.3 Random Packet Densities in Traffic

To generate packets of random sizes we have used Random Number Generators of TCL. We have modified the cbrgen.tcl for this purpose. In the code rng is a uniform random number generator declared before. We have declared a variable x_ which will have a value uniformly distributed between 0 and 3. Then is variable is used to set value of pks 128,256,384 or 512. Variable pks is used as the packet size. There are many types of random number genertors. Help for them can find in documentation of NS2.

cbrgen.tcl

```
proc create-cbr-connection { src dst }
{
.
.
.
set x_ [$rng integer 4]
set pks [expr 128 * [expr $x_ + 1 ]] ;
.
.
.
}
```

B.4 Parameters in TCL

Some times we want to pass a parameter value to a TCL script. As for example, the cbrgen.tcl script accepts rate as parameter. We would like to add another parameter mt. To do it, 1st add the code 'set opt(mt) 1' It declares that the default value of mt will be 1. There is a part in the script 'proc getopt argc argv'. This procedure will extract all the parameters. So pass the value by using $-mt$ as the last parameter. Then you can use it with the variable $\$opt(mt)$.

cbrgen.tcl

```
set opt(mt) 1
proc usage {}
{
.
.
.
     global argv0

     puts "\nusage: $argv0 \[-type cbr|tcp\] \[-nn nodes\]
\[-seed seed\] \[-mc connections\] \[-rate rate\] \[-mt max_time\]\n"
     .
     .
     .
}



proc create-cbr-connection { src dst }
{
.
.
.
global rng cbr_cnt opt defaultRNG

set stime [$rng uniform 0.0 $opt(mt)]
.
.
.
}
.
.
.
for {set i 0} {$i < $opt(nn) } {incr i}
{
.
.
.
#set dst [expr ($i+1) % [expr $opt(nn) + 1] ]//wrong
```

```
set dst [expr ($i+1) % [expr $opt(nn)] ]
.
.
.
#set dst [expr ($i+2) % [expr $opt(nn) + 1] ]//wrong
set dst [expr ($i+2) % [expr $opt(nn)] ]
.
.
.
}
```

B.5 Static Network Generation

We simulated our experiments in static environments. To accomplish this, we generated scenario files with various number of static nodes. Two parameters of "setdest" command were used for this. They are "simulation time" & "pause time". If "pause time" is greater than "simulation time", network remains static.

Appendix C

IEEE 802.11 MAC in NS2

IEEE 802.11 MAC protocol is supported in NS2. The codes for the protocol can be found in ~/ns-allinone-2.31/ns-2.31/mac/mac-802_11.cc and ~/ns-allinone-2.31/ns-2.31/mac/mac-802_11.h

C.1 Class States

- **PHY_MIB phymib_:** physical layer management information base (MIB) such as the minimal and maximal size of contention window (CWMin, CWMax), the slot time for each slot in contention wind(SlotTime), the SIFS, DIFS, PIFS, EIFS, etc.
- **MAC_MIB macmib_:** mac layer MIB such as the threshold for packet size over which the RTS/CTS would be adapted (RTSThreshold), STA short or long retry limit (ShortRetryLimit, LongRetryLimit) failure counters, etc.
- bss_id_: for network of infrastructured model.
- **basicRate_:** transmission rate for control packets such as RTS, CTS, ACK and Broadcast.
- **dataRate_:** transmission rate for mac layer data packets.
- **mhNav_:** NAV (network allocating vector) counting down timer.
- **mhRecv_:** receiving timer.
- **mhSend_:** sending timer.
- **mhDefer_:** defer timer.

- **mhBackoff_:** backoff timer.
- **nav_:** NAV in seconds.
- **rx_state_:** receiving or incoming state (MAC RECV or MAC IDLE).
- **tx_state_:** sending or outgoing state.
- **tx_active_:** transmitter is active or not.
- **Packet* pktRTS_:** outgoing RTS packet when sending RTS.
- **Packet* pktCTRL_:** outgoing control packet (CTS or ACK).
- **Packet* pktTx_:** the packet needed to be sent out such as data packet or any other packet from upper layer.
- **cw_:** current size of contention window.
- **ssrc_:** current STA short retry counter.
- **slrc_:** current STA long retry counter.

C.2 Important Functions

C.2.1 recv()

DOWN

The recv() function is called whenever a packet is received from either an upper or lower layer. The packet to be sent is received by the recv() function. The recv() function is also called when a packet comes from the channel, recv() checks the direction field in the packet header. If the direction is DOWN, meaning the packet came from an upper layer, the packet is then passed on to the send() function and return from recv().

UP

If the packet is received from a lower layer the network interface here, then the first check will be skipped. At this point the phy has just received the first bit of the incoming packet, but the MAC can't do anything with the packet until the whole packet is received. If the packet is received while the MAC is currently transmitting another packet, then the received packet would be ignored meaning the error flag in the packet's header is set. If the MAC is not currently receiving any packets, then the rx_state_ is

changed to RECV and checkBackoffTimer is called from setRxState(). Afterwards, the incoming packet is assigned to pktRx_ and the receive timer is started for the txtime() of the packet.If the MAC was already receiving a packet when this packet arrived, it will compare the received power of the new packet with the old packet. If the power of the new packet is smaller than the old packet by at least the capture threshold, the new packet will be ignored (captured) and the capture() function is called. If the power levels of the two packets are too close though, there will be a collision and control will transfer to collision(), which will drop the arriving packet. The original packet won't be dropped until it's reception is complete. Control will return to the MAC whenever the receive timer expires, calling recvHandler(), which in turns goes right to recv_timer().

C.2.2 sendDATA()

This function builds the MAC header for the data packet. First it checks that there is no packet assigned in pktTX_ variable. Then it increases the size of the packet, setting the type as data, and subtype as data. The packet should now have a complete MAC header attached to it. The function then stores the txtime of the packet, which is computed by the txtime() function. By txtime, we basically mean the size of the packet multiplied by the Data rate. This calculation is done twice this first time is just a waste. It's calculated again because a different value for the data rate is used if the packet happens to be a broadcast packet. Also, if the packet is not a broadcast packet, the duration field in the MAC header is computed. By duration, we mean the amount of time this communication still needs the channel after the data packet has been transmitted. For the case of a data packet, this corresponds to the amount of time to transmit an ACK plus a short inter-frame spacing. If the packet happens to be broadcast, this field is set to zero (no ACKs for broadcast packets). Now, the MAC has finished building the MAC header for the packet and finally assigns the internal variable pktTx_ to point to the packet we've been working on. This is essentially a way of storing the packet to be transmitted in a local buffer in the MAC. Now, the code returns to the send() function.

C.2.3 sendRTS()

This function is called from send().This function is in charge of creating an RTS packet with the specified destination in conjunction with the data packet the MAC is trying to send. The first thing it does is check the size of the packet against the RTSThreshold. If the packet is smaller (or is broadcast) then no RTS is sent before the data is transmitted (the RTS/CTS mechanism is not used). In this case, the function simply returns control back to the send() function. Otherwise, a brand new packet is created (actually done

in the first line of the function) and it's fields are set appropriately, the type is set as a MAC packet. A rts_frame structure is used to fill in the rest of the packet header and the appropriate values are put in the rts fields. The destination field is filled in with the parameter passed to the function and the rf_ra is filled in with the MAC's address. The duration field is also calculated as the time to transmit a CTS, the data packet (pktTx_) and an ACK plus 3 sifs. After the RTS has been constructed, the internal state variable pktRTS_ is assigned a pointer to the new RTS. After this, control is returned to the send() function.

C.2.4 sendACK()

This function is responsible for creating an ACK packet to be sent in response to a data packet. It checks that no other packets are previously assigned to pktTRL_ variable. The packet is created and all the fields are filled in with the obvious values. The duration field is set to zero indicating to other nodes that once this ACK has completed, they don't need to defer to another communication. Once the packet has been successfully built, pktCTRL_ is pointed to the new ACK and control returns to recvDATA().

C.2.5 deferHandler()

This function is called when the defer timer has expired. When this happens, this means the node has waited enough time before transmission to lessen the chance of collision and will now attempt to transmit a packet. Accordingly, the first thing the function does is assert that there is either a control, RTS, or data packet waiting to be transmitted. The function then calls check_pktCTRL(), and then makes sure the backoff timer is not currently running. Afterwards, it calls check_pktRTS() and check_pktTx(). If any of these check_ functions returns a value of zero, the defer handler stops, as this indicates that the check_ function has succeeded in transmitting that particular kind of packet. Therefore, the actual packet transmission is handled by one of these check_ functions.

C.2.6 check_pktRTS()

This function, like the other two check_ functions, is responsible for transmitting a packet in this case, an RTS packet. First it checks that backoff timer is not busy. If there is no RTS packet ready to send, i.e. pktRTS_ is null, then the function simply returns with a value of -1, indicating that it did not send a packet. There is an oddly placed switch statement here presumably in order to detect an improperly built RTS packet. Before the RTS is sent, the channel is checked. If it is sensed to be busy,the

contention window (cw_) is doubled using the inline function inc_cw() and the backoff timer is started again. The function therefore returns without transmitting a packet if the channel is busy. If the channel is idle, the tx_state_ of the MAC is set to RTS and the function checkBackoffTimer() is invoked. Next, the timeout value is calculated so that the MAC will know how long to wait for a CTS to be returned. Finally the function transmit() is called with arguments of the RTS packet and timeout value. At this point, the phy has begun transmission of the RTS packet.

C.2.7 check_pktTX()

This function, like the other two check_ functions, is responsible for transmitting a packet in this case, the actual data packet. The first thing the function dose is assert that backoff is not busy.If there is no data packet waiting to be sent (pktTx_ is null), then the function returns with a value of -1, indicating that nothing was transmitted. In the switch statement if the channel is sensed to be busy, sendRTS is called. This means that despite the RTS/CTS exchange, another node is using the channel (possibly due to mobility).Additionally, the contention window (cw_) is doubled using the inline function inc_cw() and then the backoff timer is started so that the MAC will remain idle until the other node has completed transmission and return. If the channel is idle, the tx_state_ is set to MAC_SEND and the checkBackoffTimer function is invoked from setTxState(). The timeout value is calculated in two ways, depending on whether or not the data packet is broadcast. If not, the timeout is how long the MAC should wait before it decides an ACK wasn't received. If the packet is broadcast, the timeout is simply the transmission time of the packet because no ACKs will be sent in conjunction with a broadcast packet. Finally, the function transmit() is invoked with arguments of the data packet and the calculated timeout value.

C.2.8 transmit()

This function takes two arguments, a packet and a timeout value. It sets a flag variable, tx_active_, to one to indicate that the MAC is currently transmitting a packet. The function then performs a check because if it is an ACK being transmitted then it is possible that the node could be receiving a packet, in which case that packet would be missed & so disarded. This next block checks if the MAC is currently receiving a packet if so, marks the packet being received as having errors. Next, the packet is actually passed down to the network interface (WirelessPhy class) which is pointed to by downtarget_. Actually, only a copy of the packet is sent down in case there needs to be a retransmission. Finally, two timers are started the send timer is started with the

timeout value, which will alert the MAC that the transmission probably failed. Also, the interface timer(mhIF_) is started with the txtime() of the packet when this timer expires, the MAC will know that the phy has completed the transmission of the packet.

C.2.9 send_timer()

This function is called at the expiration of the mhSend_. This timer expires after amount of time calculated as timeout in the corresponding check_ function the expiration of this timer means slightly different things depending on which kind of packet was sent. In a switch statement, the MAC checks the value of tx_state_ to find out the kind of packet that was most recently sent and then handles each packet differently. If the last packet sent was an RTS, the expiration of the timer means a CTS wasn't received, presumably because the RTS collided or the receiving node is deferring. The MAC responds by attempting to retransmit the RTS in the function RetransmitRTS().
If the last packet sent was a CTS packet, the expiration of the timer means that no data packet was received. This is an infrequent event occurring if the CTS packet collided or if the data packet was in error. The MAC handles this by simply resetting itself to an idle state. This involves freeing the CTS packet stored in pktCTRL_.
If the last packet sent was a data packet, the expiration of the timer means that an ACK was not received. The MAC handles this situation by calling RetransmitDATA().
Finally, if the last packet sent was an ACK, the expiration of the timer simply means that the ACK has been transmitted, as no response is expected from an ACK. The MAC frees the ACK packet pointed to by pktCTRL_ .
After each case has been handled and a packet has possibly been prepared for retransmission, the function tx_resume() is given control. If a packet is going to be retransmitted, the backoff timer has already been started with an increased contention window.

C.2.10 RetransmitData()

This function is called when an ACK is not received in response to a data packet being sent. If the data packet was a broadcast packet, an ACK shouldn't be expected and so the data packet is treated as being successfully transmitted and so is freed and the congestion window reset. The backoff counter is started. Two separate retry counts are maintained depending on whether or not an RTS is being used for this data packet. If an RTS is not being used, the short retry limit is used, otherwise the long retry limit is used as a threshold. If the retry count has exceeded the threshold, then the data packet is discarded using the discard() function and the retry count and congestion window are reset. If the retry count has not been exceeded, the data packet is prepared for

retransmission by incrementing a retry field in the mac header, doubling the congestion window, and then starting the backoff timer. This means control will eventually return to backoffHandler.

C.2.11 tx_resume()

This function is called when the MAC is getting ready to send a packet but needs to set some timers.The first thing the function does is assert that mhSend_ and mhDefer_ is not busy.If a control packet (CTS or ACK) is waiting to be sent, this function simply starts the defer time for a sifs_ amount of time. This is because a node is supposed to wait a brief period of time before transmitting. If an RTS packet is waiting to be sent, then the MAC makes sure the backoff timer isn't currently busy and bugFix_timer is false then the MAC will wait to start the defer timer. If bugFix_timer is true then backoff start with difs time. If the backoff timer isn't busy the defer timer is started for a random time in the interval [0,cw_) plus a difs_ time. If a data packet is next to be sent, and MAC isn't currently backing off, then the defer timer is started for the data packet. If an RTS wasn't used for this packet, then the defer timer is set for a random value in the interval [0,cw_] plus a difs_ time, but if an RTS was used, the MAC will only defer for a sifs_ time. This is because if an RTS was used, then the channel has already been reserved for this MAC and it shouldn't need to worry about collisions. If there are no packets waiting to be sent, but the callback_ is defined, then it is handled, corresponding to a successfully completed packet transmission. Finally, the tx_state_ is set to idle by setTxState().

C.2.12 collision()

The collision handler first checks the rx_state_ variable and sets it to MAC_COLL in case this is the first collision during the current packet. If a third packet collides, rx_state_ will already be MAC_COLL. Then, the MAC calculates how much longer the new packet will last and how much longer the old packet will last. If the new packet will last longer, then the MAC makes the new packet pktRx_ and resets the receive timer, mhRecv_. In this case the old packet is discarded here, but if the old packet will last longer then the new packet is simply discarded and pktTx_ doesn't change. So at the end of this function, the colliding packet that would have completed first has been discarded and rx_state_ is set to MAC_COLL.

C.2.13 recv_timer()

This is the receive timer handler, called when mhRecv_ expires (though indirectly though RecvHandler). The expiration of the receive timer means that a packet has been fully received and can now be acted upon.
The first thing the function does is assert that there is either a rx_state_ equals MAC_RECV or rx_state_ equals MAC_COLL.
First, the MAC checks to see if it's currently transmitting a packet by checking the flag, tx_active_. If so, the MAC wouldn't not have even heard the packet so it is just discarded (without updating NAV) and goto Done.
Next, the rx_state_ is checked to see if there was a collision during this packet, rx_state_ equals MAC_COLL. If so, then pktRx_ is the colliding packet that lasted longest and now needs to be discarded. The NAV is also set for an eifs_ time, which is the amount of time the MAC must wait after a collision and then goto Done.
The MAC then checks the packet for errors, and discards the packet if any were detected. Again, the NAV is set for eifs_ time after the error packet is finished being received.
The next check the MAC performs is if the packet is actually destined for itself if not, the MAC updates the NAV for the value in the duration field in the MAC header (not necessary just the txtime of the packet). This is of course so that the MAC doesn't attempt to transmit while other nodes are using the channel.
And finally, the last check performed is address filtering, where all packets that are not destined for the current node are discarded. The NAV would have already been updated so there's no need to do anything else with the packet.
Now the MAC decides what to do based on what kind of packet it just received. If the packet is of MAC_Type_Management, it's simply dropped. If it's an RTS packet, recvRTS() is called, if CTS or ACK, then recvCTS() or recvACK() is called. And not surprisingly, if it's a data packet, then recvDATA() is called. After this, pktRx_ is set to zero and control to given to rx_resume().

C.2.14 recvCTS()

This function is called by the recv_timer after a full CTS packet has been received, meaning the MAC can now send it's data. First it checks if the tx_state is RTS, if not we should not have received CTS so the packet is discarded & the function returned.Since the MAC has no use for the RTS packet it just transmitted, it's freed and pktRTS_ is set to zero. The send timer is stopped. Control then goes straight to tx_resume(), which sets the defer timer, and then control finally returns back to recv_timer().

C.2.15 recvACK()

This function is called by the recv_timer after a full ACK packet has been received, indicating a successful data transmission. First, the MAC checks that it really did just sent a data packet (tx_state == MAC_SEND) and discards the ACK if it didn't. Then the send timer is stopped The MAC now knows that it just succesfully transmitted it's data packet, so it frees pktTx_ and sets it to zero.The MAC then resets the appropriate retry count, short if an RTS wasn't used, long if it was. Also, the congestion window is reset and the MAC starts its backoff timer so it won't just immediately send again. Control then goes to tx_resume() and then back to recv_timer(). In tx_resume(), since there are no packets ready to send, the callback will be invoked, effectively telling the interface queue to send down another packet for transmission.

C.2.16 rx_resume()

This simple function is called after recv_timer has completed. All it does is set the rx_state_ to idle which eventually invokes checkBackoffTimer().

C.2.17 backoffHandler()

This function is called whenever the backoff timer expires. This function first checks to see whether there is a control packet (CTS or ACK) waiting to be sent. If so, it makes sure that the MAC is either sending the packet or deferring before sending the packet. If there was no control packet, check_pktRTS() is called. If there was no RTS packet, then check_pktTx() is called. This means, that at the expiration of the backoff timer, an RTS or a data packet will be transmitted if either is waiting.

C.2.18 Miscellaneous Functions

- **set_nav(u_int16_t):** set the NAV according to the given unsigned short integer value, times s; if the existing NAV is later than this value, ignoring it, otherwise, update the NAV;
- **is_idle(void):** checking the receiving state, sending state and NAV, if any of them is not idle,then the state of MAC is not idle;
- **inc_cw():** increasing number of slots in the contention window when backoff; IEEE 802.11 standard specifies it from 63 to 1023, each time increasing by an order of 2;

- **rst_cw():** reseting the contention window to the basic setting, 63 slots;
- **sec(double):** given an integer value of s, return its equivalent value in seconds;
- **usec(double):** given a double value of seconds, return its integer value of s
- **txtime(Packet*):** return the transmission time eld in common header hdr_cmn::txtime();

C.3 Timers

Timers are defined in the files mac/mac-timers.h/cc while the handlers (functions called when the timer expires) are in mac-802_11.cc.

- **IFTimer** The interface timer keeps tracks of how long the interface will be in transmit mode. This is only the time when the interface is actively transmitting bits into the air. The handler for this timer is txHandler(). Probably the simplest timer used by the MAC layer.
- **NavTimer** Started at the reception of a packet for the length of time indicated in the duration field of the MAC header. Most 802.11 frames carry a duration field, which can be used to reserve the medium for a fixed time period. The NAV is a timer that indicates the amount of time the medium will be reserved, in microseconds. Stations set the NAV to the time for which they expect to use the medium, including any frames necessary to complete the current operation. Other stations count down from the NAV to 0. When the NAV is nonzero, the virtual carrier-sensing function indicates that the medium is busy; when the NAV reaches 0, the virtual carrier-sensing function indicates that the medium is idle. Calls navHandler() on expiration.
- **RxTimer** Started when the first bit of a packet is received and set for the length of time the packet will require to be completely received. This timer is needed because in simulation the entire packet is available as soon as the first bit arrives, but the MAC should not access the packet until it would have been completely received in reality. In the case of a packet collision, the receive timer is reset to expire at the end of the last colliding packet. The timer indirectly calls recv_timer() on expiration by calling recvHandler() first.
- **TxTimer** Indicates the time by which ACK/CTS should have been received. The TxTimer (mhSend_) is started when a packet is transmitted by the transmit() function. Each type of packet has an expected response, for example, an RTS packet expects a CTS packet to follow. The timer is therefore stopped when a

CTS, data, or ACK packet is received. The timer is not started on transmission of an ACK packet as there is no response expected. On expiration, send_timer() is called indirectly by first calling the function sendHandler().

C.4 Flow of Transmission

Node A tries to send a packet to Node B. Follow each tree in Depth First Order to get the flow.

C.4.1 Successfull Transmission

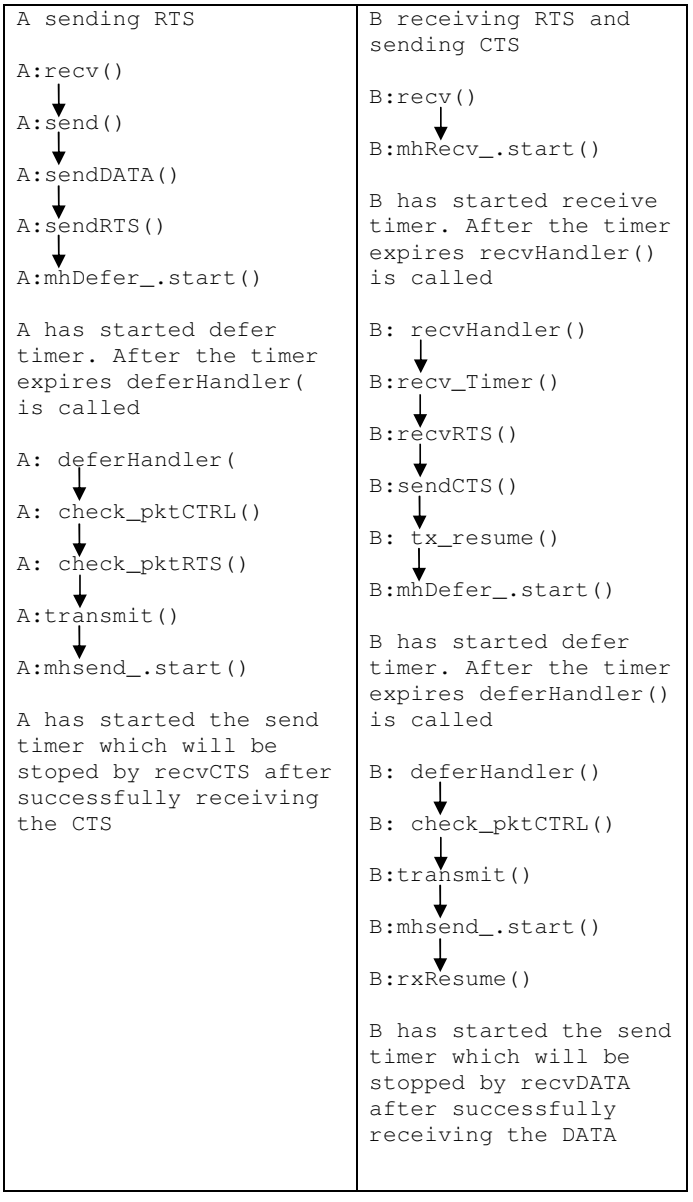

FIGURE C.1: Successfull transmission: Part1

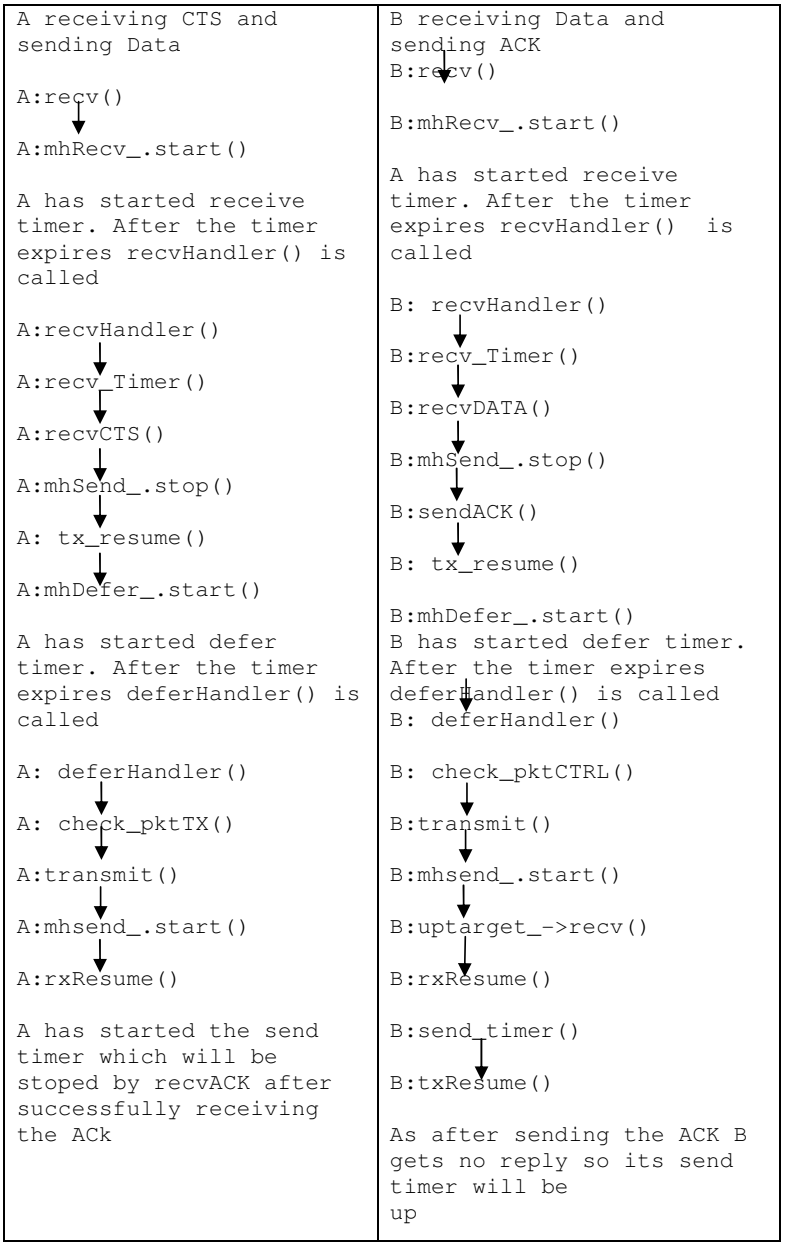

FIGURE C.2: Successfull transmission: Part2

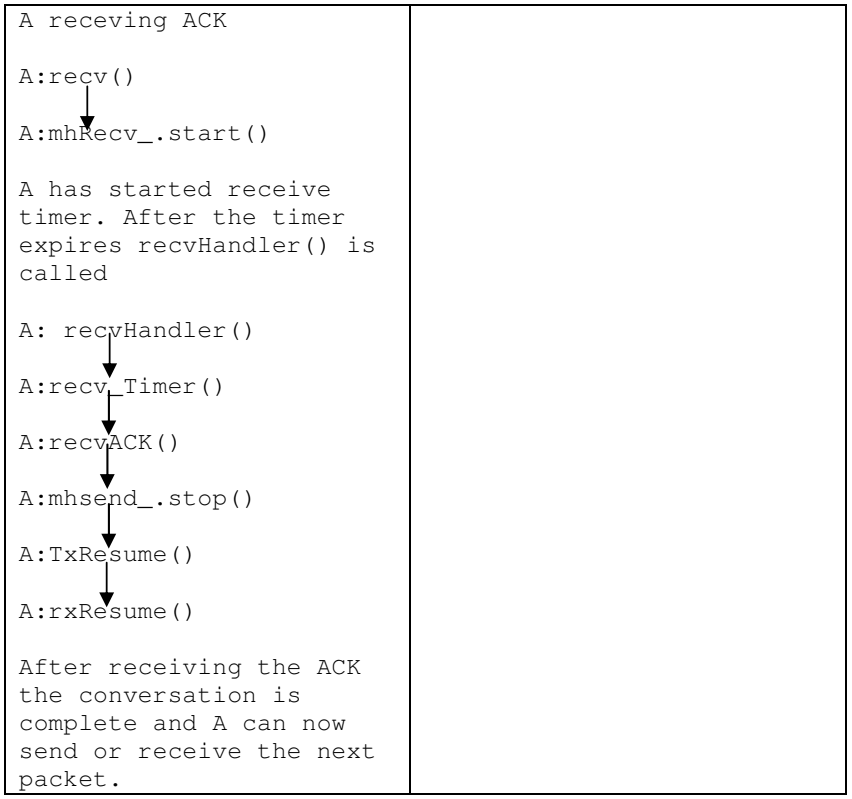

FIGURE C.3: Successfull transmission: Part3

C.4.2 RTS RE-Transmission

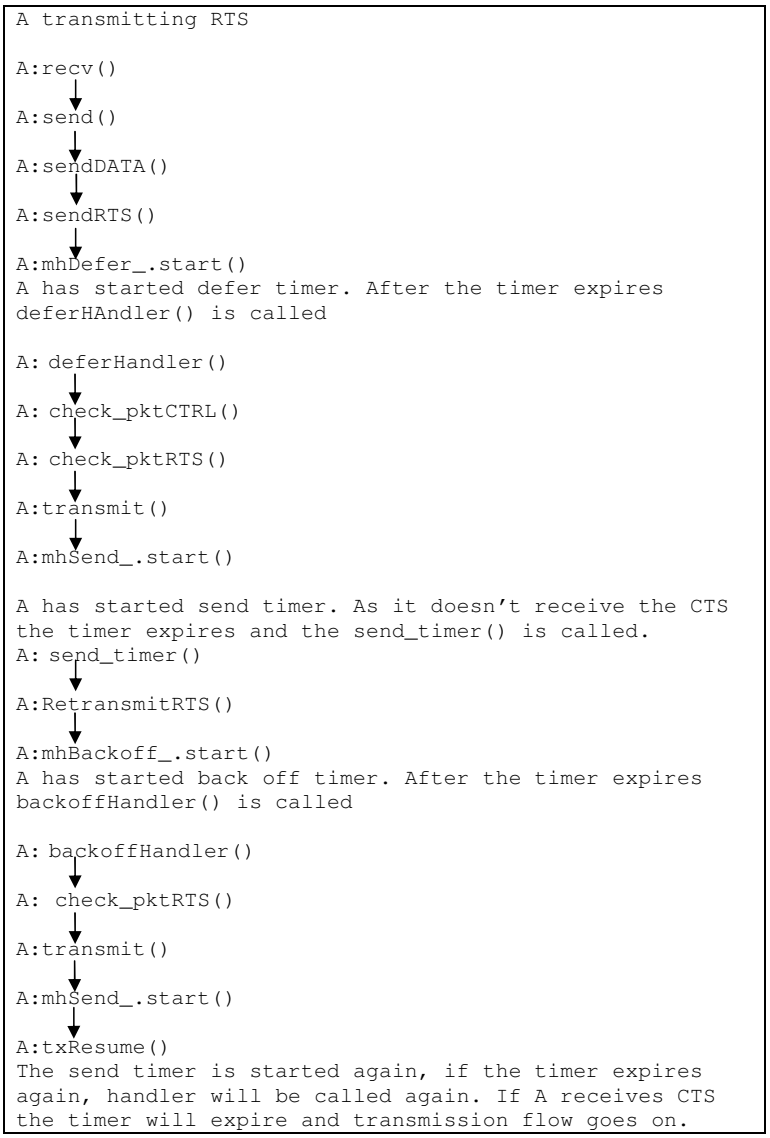

FIGURE C.4: RTS Retransmission

C.4.3 Data RE-Transmission

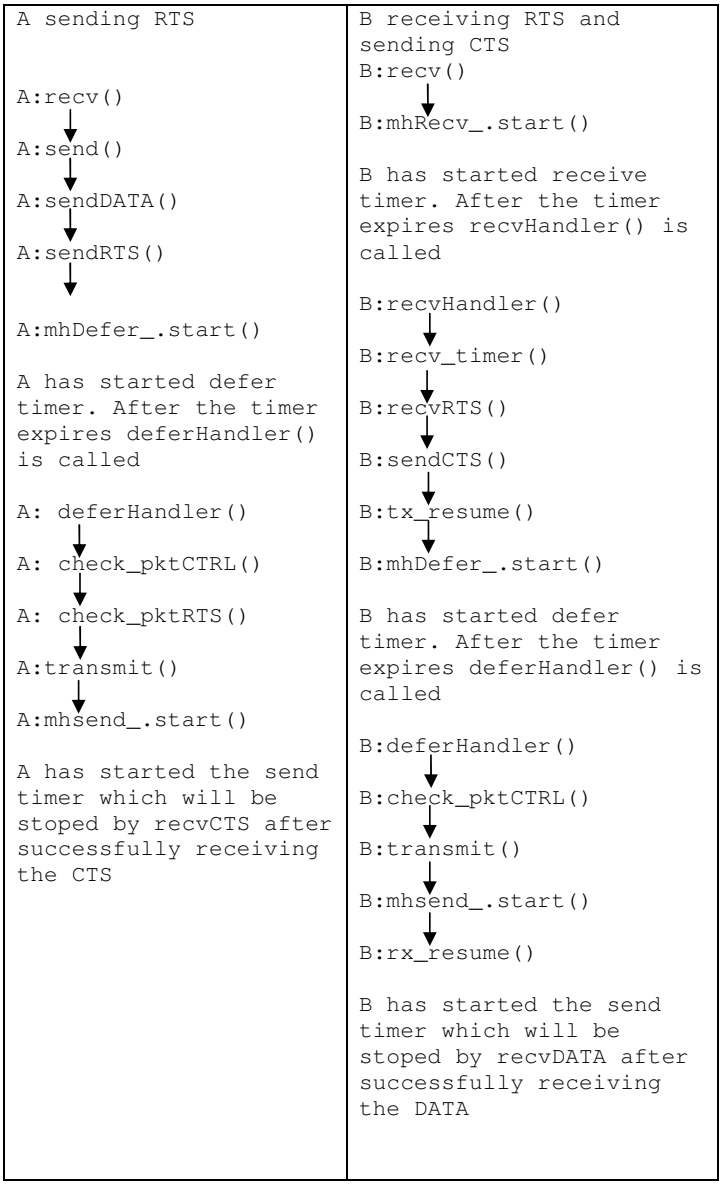

FIGURE C.5: Data Retransmission: Part1

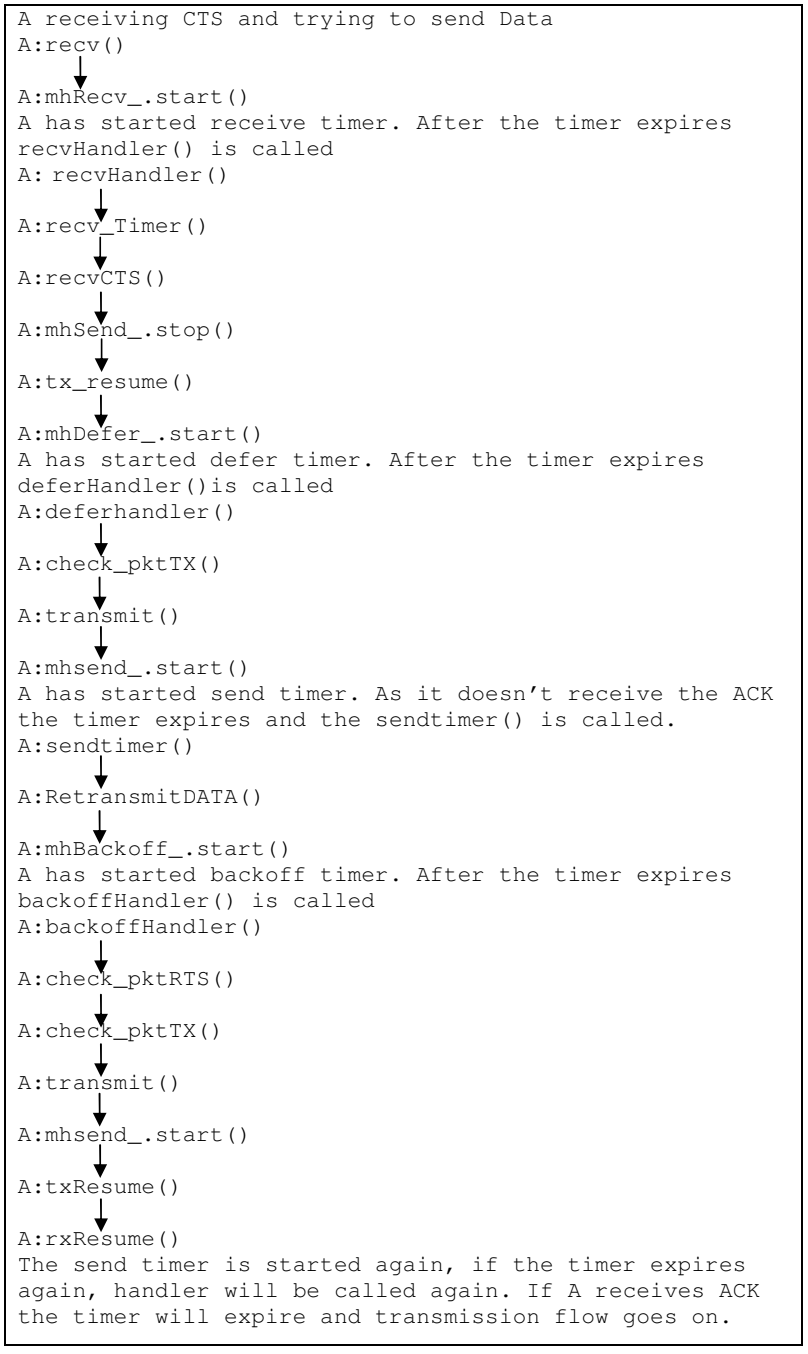

FIGURE C.6: Data Retransmission: Part2

Appendix D

GNUPlot

To visualize our simulation results and compare proposed adaptive RTS Threshold scheme with existing static mechanism, we used Gnuplot as an interactive plotting program which helped us with its manifold apts. Gnuplot version 4.2 was available and free to use. Below is a brief description of this software.

D.1 Description

Gnuplot is a command-driven interactive function plotting program. If files are given, gnuplot loads each file with the load command, in the order specified. Gnuplot exits after the last file is processed. Here are some of its features are,

1. Plots any number of functions, built up of C operators, C library functions, and some things C doesnt have like **, sgn(), etc. Also support for plotting data files, to compare actual data to theoretical curves.
2. User-defined X and Y ranges (optional auto-ranging), smart axes scaling, smart tic marks.
3. Labelling of X and Y axes.
4. User-defined constants and functions.
5. Support for many output devices and file formats
6. Shell escapes and command line substitution.
7. Load and save capability.

8. Output redirection.

All computations performed in the complex domain. Just the real part is plotted by default, but functions like imag(), abs() and arg() are available to override this.

Example:

- set x-axis label: set xlabel "X_AXIS_LABEL"
- set y-axis label: set ylabel "Y_AXIS_LABEL"
- set legend position: set key [left/right top/bottom]
- set for greek letters: set terminal postscript eps enhanced
- set output file: set output "FILE.eps"
- plot from a file: plot "FILE" with [lines/points/linespoints]
- plot with different ratio(for example 2:3): plot "FILE" using ($1 × 2) : ($2 × 3) with lines
- put greek letter in text: ".../Symbol a..." for alpha, ".../Symbol h..." for eta

Helping Site: http://t16web.lanl.gov/Kawano/gnuplot/index-e.html

Bibliography

[1] Darpa home page. URL http://www.darpa.mil.

[2] Ietf manet working group information. URL www.ietf.org/html.charters/manet-charter.html.

[3] Y.D. Lin and Y.C. Hsu. Multi-hop cellular: A new architecture for wireless communications. In *IEEE INFOCOM*, pages 1273–1282, 2000.

[4] R. Pickholtz A.N. Zadeh, B. Jabbari and B. Vojcic. Self organizing packet radio ad hoc network with overlay. *IEEE Communications Magazine*, 40(6):140–157, June 2002.

[5] P. Chatzimisios and A. C. Boucouvalas. Improving performance through optimization of the RTS/CTS mechanism in IEEE 802.11 wireless lans. In *International Conference on Communication Systems, Networks and Digital Processing (CSNDSP 2004)*, pages 375–378, 2004.

[6] G. Bianchi. Performance analysis of the IEEE 802.11 distributed coordination function. *IEEE Journal of Selected Areas in Communications*, 18(3):535–547, 2000.

[7] A.C. Boucouvalas P. Chatzimisios and V. Vitsas. Packet delay analysis of the IEEE 802.11 MAC protocol. *IEEE Electronic Letters*, 39(18):1358,1359, 2003.

[8] S. Xu and T. Saadawi. Revealing the problems with 802.11 medium access control protocol in multi-hop wireless ad hoc networks. *Computer Networks*, 38(4):531–548, 2002.

[9] M. Gerla K. Xu and S. Bae. Effectiveness of RTS/CTS handshake in IEEE 802.11 based ad hoc networks. *Ad Hoc Networks Journal*, 1(1):107–123, 2003.

[10] J.-H. Chen F.Ye S.-T. Sheu, T. Chen. The impact of rts threshold on IEEE 802.11 mac protocol. In *IEEE International Conference on Parallel Distributed Systems (ICPADS)*, pages 267–272, 2002.

[11] Shiqi Wu-Wei Guo Shaohu Yan, Yongning Zhuo. Adaptive RTS threshold for maximum network throughput in IEEE 802.11 DCF. volume 5284, pages 332–343, 2004.

[12] Bai long Xiao Fei Huang Jun Liu, Wei Guo. RTS threshold adjustment algorithm for IEEE 802.11 DCF. In *6th International Conference on ITS Telecommunications*, pages 654–658, 2006.

[13] Naoki Nakamura Kazuhide Koide Atushi Takeda Norio Shiratori Mostafa Mjidi, Debasish Chakraborty. A new dynamic scheme for effcient RTS thresholdhandling in wireless networks. In *22nd International Conference on Advanced Information Networking and Applications*, pages 734–740, 2008.

[14] Yen-Cheng Kuan Huei-Jiun Ju, Izhak Rubin. An adaptive RTS/CTS control mechanism for IEEE 802.11 MAC protocol. *Vehicular Technology Conference, 2003. VTC 2003-Spring. The 57th IEEE Semiannual*, 2:1469– 1473, 2003.

[15] P. Karn. MACA-a new channel access method for packet radio. In *9th Computer Networking Conference on ARRL/CRRL Ameture Radio*, pages 134–140, 1990.

[16] Vaduvur Bharghavan, Alan Demers, Scott Shenker, and Lixia Zhang. MACAW: A media access protocol for wireless lan's. In *ACM SIGCOMM '94*, pages 212–225, 1994.

[17] Ashikur Rahman and Pawel Gburzynski. Hidden problems with the hidden node problem. In *23rd Biennial Symposium on Communications- QBSC 2006*, pages 270–273, 2006.