SERVICE DIFFERENTIATION USING

p-PERSISTENT CSMA/CA

By

VIJAY GURUSAMY

Bachelor of Engineering in Computer Science

Bharthidasan University

Trichy, India

2002

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2005

SERVICE DIFFERENTIATION USING

p-PERSISTENT CSMA/CA

Thesis Approved:

Dr. Venkatesh Sarangan

Thesis Advisor

Dr. John P Chandler

Dr. N. Park

Dr. A.Gordon Emslie

Dean of the Graduate College

ACKNOWLEGEMENTS

I wish to express my sincere gratitude to my advisor, Dr. Venkatesh Sarangan, for his guidance, assistance, and motivation throughout the study. I am very grateful to him for his persistence and support in completing my thesis work.

My sincere appreciation extends to my committee members, Dr. John P. Chandler and Dr. N. Park for their suggestions and supervision.

I would like to thank my mother Mrs. Saroja Gurusamy, father Mr. Gurusamy and brother Mr. Ramanujam for their love and support throughout the years.

Finally, I would like to thank all my friends who stood beside me when I was down at times with their constant moral support.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

1.1 Wireless Local Area Networks

Wireless networking and multimedia are two fast growing technologies. Wireless Local Area Network (WLAN) is a flexible data exchange system that can either add functionality to a wired network or replace the existing one. A WLAN has the luxury of not being connected by a cable as well provides all the features and benefits of traditional LAN technologies like Ethernet and Token Ring [1].

Temporary installations represent one situation of when a wireless networks might make sense or even is required. The increasing number of mobile users is a clear candidate for WLAN. Portable access to WLANs can be achieved using notebook computers and wireless Network Interface Cards. This makes the users travel to various locations like meeting rooms, hallways, lobbies, cafeterias, classrooms, etc. and still have access to their networked data. Without a WLAN, the user has to carry an awkward cable and find a network tap to plug into.

In all these scenarios it is worth mentioning that today's standards-based WLANs operate at high speeds – the same speed which where considered state of art for wired networks a few years ago. There are numerous WLAN solutions available today, with varying levels of standardization and interoperability. The solution that is currently leading the industry is Wi-Fi^{TM} (IEEE 802.11b). WLANs are built using two basic topologies. They are as follows:

1) Infrastructure: This topology can extend wired LAN to wireless devices by providing a base station called an access point. The access point acts as a central coordinator connecting the wired network and wireless network. In this mode there could be multiple access points to cover a large area or only a single point for a small area or small building.

2) Ad-hoc: In this topology, the wireless devices themselves, with no central coordinator like an access point create a WLAN. Each device communicates with other devices in the network rather than through a central coordinator.

As WLAN is a new networking medium, which has to face all the new challenges that arises when introduced into a new environment. Considering the challenge of transmitting multimedia contents had taken its toll on the technology. Real-time and multimedia applications require some Quality of Service (QoS) support such as guaranteed bandwidth, bounded delay and jitter. Providing such QoS support in 802.11 is challenging since 802.11 does not take QoS support into account [3, 4]: both the Medium Access Control (MAC) layer and the Physical (PHY) layer are designed for best-effort data transmission.

The IEEE 802.11 MAC specifies two different MAC mechanisms in WLANs: the mandatory contention based Distributed Coordination Function (DCF) and the optional polling based Point Coordination Function (PCF) [3].

1.2 Distributed Coordination Function (DCF)

DCF is a mandatory asynchronous mechanism. Before a station starts transmission, it senses the wireless medium to determine whether it is idle. If the medium is idle, station defers its transmission until the ongoing transmission is completed. The CSMA/CA mechanism requires a minimum time interval between contiguous frame transmissions. A station will ensure that the medium is idle for the specified interval of time before attempting to transmit.

1.3 Point Coordination Function (PCF)

PCF is an optional synchronous mechanism which implements polling based contention-free access scheme. It can be only used with an infrastructure mode. The reason is that PCF relies on asynchronous service provided by DCF mechanism, which should at least send one DCF data frame in a beacon interval. Moreover, PCF uses a centralized polling scheme, which uses the access point (AP) as a point coordinator (PC).

This thesis aims to provide novel approach for providing service differentiation using p-Persistent CSMA/CA logic. The following chapter 2 gives the background regarding this thesis. Chapter 3 provides the problem statement. Literature review is done in chapter 4. Chapter 5 illustrates the proposed solution. Simulation plan is discussed in chapter 6.

CHAPTER 2

BACKGROUND

This chapter discusses all the fundamentals of wireless local area networks and CSMA protocols along with all the variants of the protocols.

2.1 What is CSMA protocol ?

Carrier Sense Multiple Access (CSMA) Protocols are protocols in which the station listen for a carrier (or a transmission) and act accordingly.

There are several variants of CSMA contention protocols.1-persistent CSMA, non-persistent CSMA, and p-persistent CSMA, are the various versions of the carrier sense protocol. The following paragraphs discusses about the variants:

2.2 One-Persistent CSMA

When a station has data to send, it first listens to the channel to see if any station is transmitting at that moment. If the channel is busy, the station waits until it becomes idle. When the station detects an idle channel, it transmits a frame. If a collision occurs, the station waits a random amount of time and starts all over again. This version of CSMA transmits with a probability of 1 whenever it finds the channel idle.

2.3 Non-Persistent CSMA

In this version, an attempt is made to make the 1-persitent CSMA version less greedy. Before sending, a station senses the channel. If the medium is idle, the station begins transmitting. However, if the channel is already in use, the station does not continuously sense to seize the channel upon detecting the end of previous transmission, instead it waits a random period of time and then repeats the whole process. This algorithm should lead to better channel utilization and longer delays than 1-persistent CSMA.

2.4 p-Persistent CSMA

This version of CSMA applies to slotted channels. When a station becomes ready to send, it senses the channel. If it is idle, it transmits with a probability p. With a probability q=1-p it defers until the next slot. If that slot is also idle, it either transmits or defers again, with the probabilities p and q. The process is repeated until either the frame has been transmitted or another station has begun transmitting. If another station has begun transmitting, it acts as if there had been a collision, which means that it waits a random time and starts again. If the station initially senses the channel busy, it waits until the next slot and applies the whole process again.

2.5 IEEE 802.11 DCF

The DCF [2] achieves medium sharing between compatible stations through the use of Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA). Collision detection cannot be implemented due to the significant difference between transmitted and received power levels. Two different carrier-sensing mechanisms are used [8]: PHY carrier sensing at the air interface and virtual carrier sensing at the MAC layer. By analyzing all the packets received from other stations (STA), the PHY carrier sensor can detect the presence of other STAs. A station to inform all other stations how long the channel will be reserved for transmission optionally uses virtual carrier sensor. In order to avoid this scenario, the sender can set a duration field in the MAC header of data frames, or in the RequestToSend (RTS) and ClearToSend (CTS) control frames. Accordingly, other stations will update their local timers of network allocation vectors (NAVs) to take the duration into account.

As shown in figure 1 [8], if a packet arrives at an empty queue and if the medium has been found idle for more than distributed interfame space called DIFS, the source station can transmit the packet immediately. In the meantime, rest of the stations defer their transmissions by adjusting their respective NAVs, and start the backoff process. Stations compute their backoff timer, which is a random time interval selected from the Contention Window (CW): backoff timer = random [0, CW].slot time, where slot time depends on the PHY layer type and $CW_{min} < CW < CW_{max.}$ The backoff timer parameter is decreased until the medium is idle. It is frozen once the medium is busy.

To reduce probability of collisions, after each unsuccessful transmission attempt, the CW value is doubled and after every successful transmission attempt the CW value is reset to $CW_{min.}$ According to the standard, a maximum of 7 retransmissions for short frames are allowed before the frame is dropped. Hidden terminals can also cause collisions. Consequently, frames sent from different senders will collide at the same receiver. To solve the hidden terminal problems RTS/CTS mechanism can be used optionally. As shown in figure 1 [8], the source sends a short request to send (RTS) frame before each transmission begins.

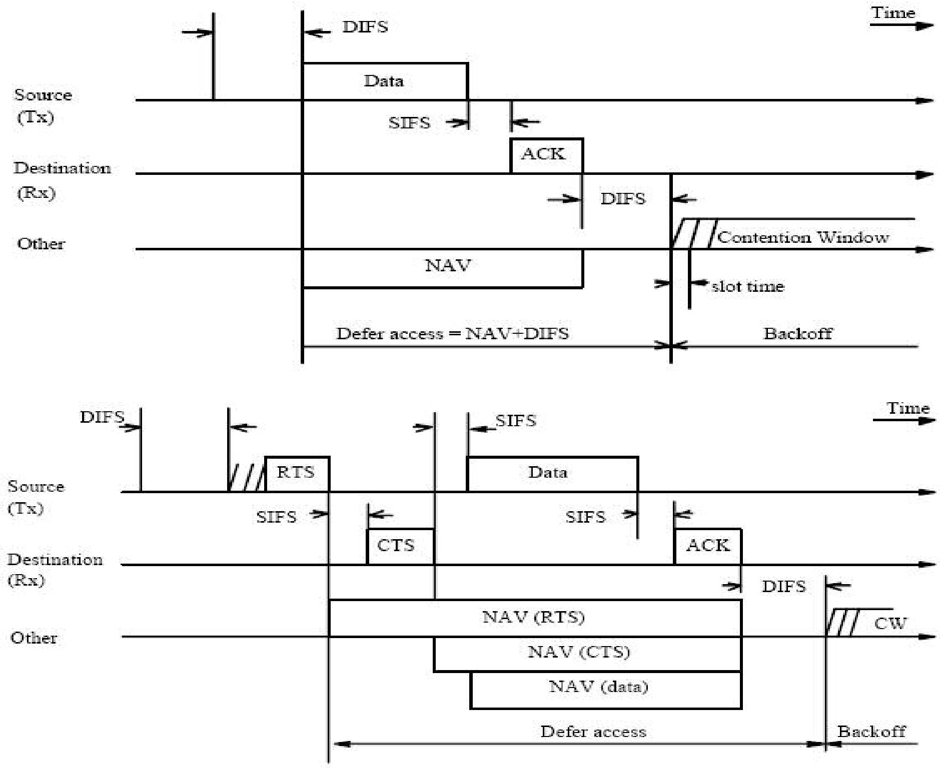

Figure 1: DCF Access Mechanism: CSMA/CA (up) and RTS/CTS scheme (down) [8]

The receiver replies with a clear to send (CTS) frame if it is ready to receive. Once the source receives the CTS frame, it transmits a frame. All other station in the same BSS hearing the CTS frame adjusts the respective network allocation vectors (NAVs). The rest of the stations in the BSS will not attempt to start transmission until the NAV timer reaches zero. If the data frame sizes are large, the RTS/CTS mechanism can improve the performance significantly.

2.6 IEEE 802.11 PCF

Priority-based access can also be used to access the medium. Unlike DCF, its implementation is not mandatory. The reason is the implementation of PCF itself was thought to complex and not finalized in the standard. PCF uses a centralized polling scheme, which uses the access point (AP) as a point coordinator (PC). When a BSS is set up with a PCF-enabled, the channel access time is divided into periodic intervals named beacon intervals. As shown in the figure 2 [8], the beacon interval is composed of a contention-free period (CFP) and a contention period (CP).

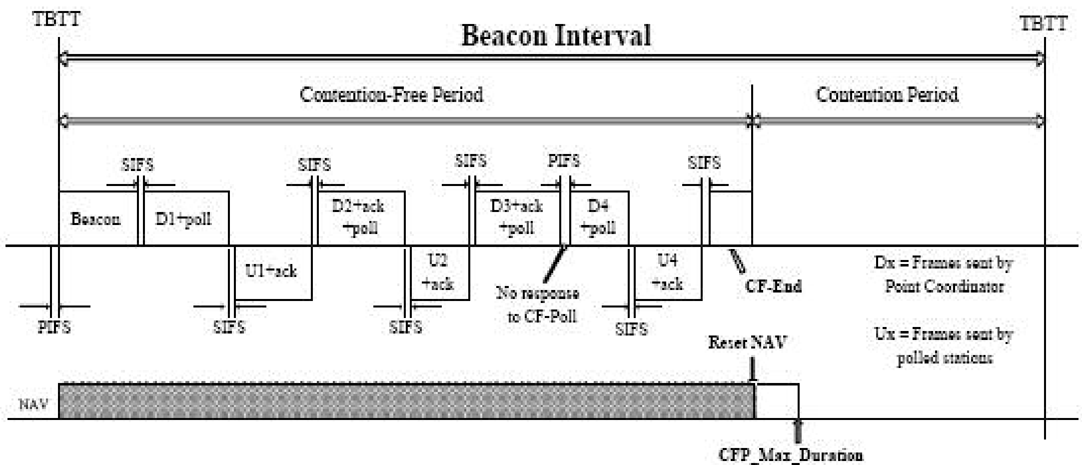

Figure 2: PCF and DCF alternation [8]

During the CFP, the PC maintains a list of registered stations and polls each of them according to the list. When a station is polled, it starts to transmit data frames, where the size of each frame depends on the maximum MAC service data unit size. The time used by the PC to generate the beacon frame is called target beacon transmission time (TBTT). The next TBTT is sent within the beacon frame by the PC to inform all other stations in the BSS. To give higher priority of access than DCF in a beacon interval, the PC waits for a shorter interframe space than DCF interframe space (DIFS). PCF is not allowed to interrupt any ongoing frame transmissions in DCF. Once PCF obtains access to the wireless medium, short interframe space (SIFS) timing is used for frames exchanges during CFP except if the polled station does not respond the PC within a PIFS period. PCF and DCF cycles take place as shown in the figure 2 [8]. If the stations have no data to transmit during CFP, the AP can terminate the CFP by sending CF-end frame. After

CF-end frame, the CP is started again and it will remain so until a CFP AP transmits starting beacon. In general, PCF uses a round-robin scheduler to poll each station sequentially in the order of the polling list. The PCF is used for delay sensitive data to meet their delay requirements.

2.7 Need for QoS enhancement

Maintaining QoS is one of the most challenging functions a MAC layer should support. QoS is the ability of a network element to provide some levels of assurance for consistent network data delivery [8]. When a bandwidth is not scarce, such as in wired LANs, QoS issues are not so important. However, a WLAN have a higher bit error rate, a higher delay and lower bandwidth than a wired LAN. IEEE 802.11 WLAN is originally designed for best-effort services. The error rate at physical layer is more than three orders of magnitude larger than that of wired LAN. Moreover, high collision rate and frequent retransmissions cause unpredictable delays and jitters, which degrade the quality of real-time voice and video transmission. Enhanced QoS aware coordination can reduce overhead, prioritize frames, and prevent collisions to meet delay and jitter requirements in mobile environments.

CHAPTER 3

HYPOTHESIS

QoS can be provided using persistent CSMA logic. Basically, CSMA/CA protocol supports that all stations has the same priority. The p-persistent CSMA logic can be used to achieve QoS among the various WLAN stations. In the following paragraphs we formulate the problem of providing priorities to various WLAN stations involved in the wireless network. The problem here is to identify the stations with higher priority and lower priority. Usually, the priorities to all the stations in a p-persistent CSMA/CA logic is maintained the same by having a common 'p' value for all the stations.

Hence, the question arises how to distinguish a high priority station from the low priority station?

The problem can be analyzed as follows. Consider a situation where a particular station 'A' has real-time applications like voice or live video transmission. While another station 'B' which has a data transmission application. In this scenario, the station 'A' has to be given a higher priority since there are voice and video packets to be transmitted.

One way to provide priority is by statically fixing the stations with a constant 'p' value. According to the rule of probability, higher the value of 'p' there are more chances of packet transmission and vice versa. This concept is explained in detail in the following proposed approach.

CHAPTER 4

LITERATURE REVIEW

Reference [8] presents a review of QoS enhancement research efforts and standardization activities of wireless LANs. It also discusses about the various QoS enhancement schemes, which are classified into station-based and queue-based categories. A station-based category will provide only one priority for one station and queue-based category will introduce multiple priority queues in each station. Another level of classification depends on whether the scheme is DCF-based or PCF-based.

4.1 IACC Scheme

IACC scheme introduce priorities using three techniques. The first one allocates different contention windows to stations with different priorities. Experiments shows that this technique performs well with UDP traffic were as performs badly with TCP. In the second technique, each station sets the DIFS parameter according to its priority level. The problem with this technique is the low priority traffic suffers from starvation. While the third technique, each station is allocated a different maximum frame length according to the priority level. This technique seemed to work for TCP and UDP flows but in a noisy environment it tends to decrease the efficiency of service differentiation.

4.2 Blackburst Scheme

The objective of the Blackburst scheme is to minimize the delay of real-time traffic. Unlike other schemes, it imposes certain requirements on high priority stations: (1) the use of equal and constant intervals to access the medium and (2) the ability to jam the medium for a period of time. If there are no constant access intervals for high priority station, performance of the Blackburst scheme degrades considerably.

4.3 JDRC Scheme

In the JDRC scheme, high priority stations will be able to access medium with short waiting time. On the downside, when none of the high priority station wants to transmit the low priority station still have a longer waiting time.

4.4 Hybrid Coordination Function Scheme

Reference [5] discusses about the IEEE 802.11e Medium Access Control, which emerged as a standard to support QoS. The Hybrid Coordination Function (HCF) in 802.11e is only efficient for flows with strict Constant Bit Rate (CBR) characteristics. The HCF access method is a combination of the EDCF mechanism and Hybrid Controlled Channel Access mechanism. However, a lot of real-time applications such as video conferencing have small variations in packet sizes, which leads to Variable Bit Rate (VBR) characteristics. Hence, priority to real-time traffic cannot be supported which needs a better algorithm to distinguish the stations need for higher priority.

Reference [13] talks about a new proposed standard IEEE 802.11e, which supports QoS in wireless LANs. It introduced a new access method Hybrid Coordination Function (HCF) that combines the DCF and PCF mechanisms.

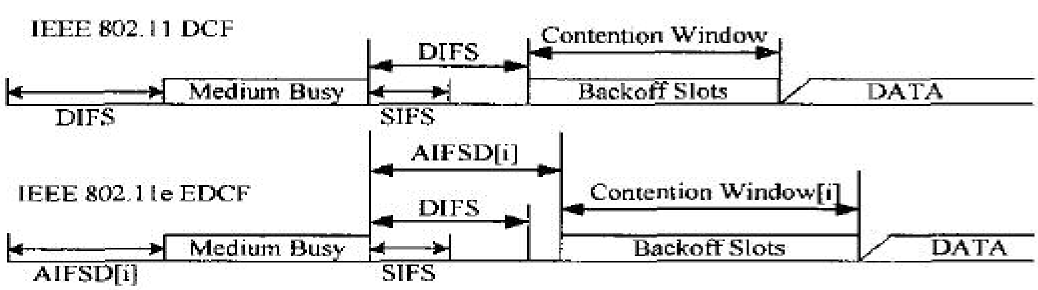

Figure 3: Comparison of 802.11 and 802.11e EDCF [15]

Enhanced DCF is a contention-based HCF access method specified in 802.11e. An 802.11e QoS enhanced station can support at most four access categories (AC) of which it can support eight types of traffic classes. Each traffic type is mapped into particular access category with the priority range from 0 to 7 [14]. As shown in figure 3, in IEEE 802.11e, a QoS enhanced station uses a new Arbitration Interfame Space, AIFSD [AC], CW_{min}[AC] and CW_{max}[AC] instead of DIFS, CW_{min} and CW_{max} of the DCF. The respective AC is selected according to the priority range selected for that application. By differentiating the inter frame space time and contention window of each access category, the traffic with high priority will have more chances to gain the channel access and suffer less delay.

4.5 Adaptive Service Differentiation Scheme

Romdhani, Ni and Turletti [7] proposed an adaptive service differentiation scheme for QoS enhancement in IEEE 802.11e. The contention window (CW) parameter is set statically in Enhanced DCF (EDCF) were as in this scheme CW value is set dynamically. They use a dynamic procedure to change the CW value after each successful transmission or collision. Though, the performance of this scheme is better than EDCF the background performance of the low priority flows degrades at high loads.

4.6 Adaptive fair EDCF Scheme

Malli, Ni, Turletti and Barakat [6] suggested a new scheme called adaptive fair EDCF that extends EDCF, which combines the advantages of service differentiation, fast backoff decrease and an adaptive access scheme (using an adaptive backoff threshold). This scheme aims to improve the performance of multimedia applications, total throughput and fairness between same priority applications.

4.7 Dynamic Tuning of IEEE 802.11 protocol

Cali, Conti and Gregori [10], [11], [12], are the first to derive analytic models that characterize the system capacity using the p-persistent version of IEEE 802.11. The capacity is defined as the maximum fraction of the channel bandwidth used for successful packet transmission. However, they consider only a single class of traffic that does not address the issue of providing differentiation among multiple traffic classes. We extend their work, and propose to use p-persistent CSMA logic to introduce priority among different class of stations.

CHAPTER 5

PROPOSED SOLUTION

5.1 Introduction

This chapter explains the solution for the proposed problem. A p-persistent CSMA version of the IEEE 802.11 protocol differs from the standard protocol only in the selection of the backoff interval. In the standard version of p-persistent CSMA logic, the ‘p’ value remains a constant for all the stations in the wireless network.

5.2 Priorities Assignment

The priorities to various stations are assigned after analyzing the applications that is running on respective stations. The station that has real-time application such as multimedia contents to transmit is given higher priority over the other applications, which transmits data contents.

5.3 Service Differentiation Rule

A constant probability value ‘p_1’ is fixed for a higher-class priority station. Similarly, a value ‘p_2’ is fixed for a lower-class priority station. The constant value for each station that is assigned follows a rule; higher-class priority value should be greater than lower-class priority value (i.e. $p_1 > p_2$). Since the fixed values a probability values they are assumed to be in the range 0 to 1.

We have to distinguish the class-level station priorities using the following method:

i) A random value ‘p’ is generated between 0 and 1

ii) If $p < p_i$ Then

Respective station is given access to transmit

Else

The transmission is deferred to the next slot

In the above process, 'p_i' denotes the probability that is fixed for the different class-level priorities and the subscript 'i' refers to the various class-levels.

Depending on the random value of 'p' that is generated there could be two different cases, which are discussed as follows:

Case I:

Suppose, if the random 'p' value is lesser than the fixed probability of the respective class-level priority then the station, which belongs to that particular class-level priority is given access to transmit the packets.

Case II:

Suppose, if the random 'p' value is greater than fixed probabilities of both class-level priorities then the access to the channel is deferred to the next slot with a probability of '1-p'.

For example, consider a scenario where there is a two class of stations 'A' and 'B'. If class 'A' stations has real-time applications then it is assigned a high priority like $p_1=0.8$. If class 'B' stations contains only data contents will be assigned a low priority like $p_2=0.3$. The above algorithm is executed on each class of machines individually.

Case 1:

Suppose, if the random 'p' value generated is 0.2 that is less than 0.3 (class 'B')

Then class ‘B’ stations is given access to transmit

Case 2:

Suppose, if the random ‘p’ value obtained is 0.7 that is less than 0.8 (class ‘A’)

Then class ‘A’ stations is given access to transmit

Case 3:

Suppose, if the random ‘p’ value obtained is greater than probabilities of either class of stations ‘A’ or ‘B’

Then transmission is deferred with probability of ‘1-p’

The proposed approach is a new approach to the problem. It is a well-organized and simple approach to the problem. Unlike the existing p-persistent CSMA version of the protocol, this approach addresses the issue of class priority for an individual station. This solution can be implemented easily with minor modifications to the p-persistent CSMA version. Furthermore, it can be deployed across the wireless networks without any major changes in the infrastructure.

CHAPTER 6

SIMULATION

In the simulation, we try to study the behavior and performance of the algorithm discussed in the previous chapter. We implemented the modified p-persistent CSMA version in the wireless environment using the OPNET modeler.

6.1 Introduction

I chose the OPNET simulator because it has a lot of modules that is needed for this simulation. OPNET Modeler [16] supports all network types and technologies. Among the many benefits of this development environment are: its hierarchical network models, its clear modeling paradigm, its finite state machine design capabilities, its integrated analysis tools, its comprehensive libraries of protocol, application, and network devices, its wireless, point-to-point, and multilink functionality. The wireless WLAN module of the simulator was used as the basis for modifying and implementing the class priority using the p-persistent CSMA/CA version.

6.2 OPNET Implementation

The p-persistent CSMA/CA version was implemented and integrated into the wlan_mac process model as shown in the figure 4. One transition and one condition was inserted into the process model and the TRANSMIT state was modified. I also modified the interrupts in the Function Block and added a priority to the Node Attributes Interface for easy assignment of class priorities for various stations.

The additional pseudo code for TRANSMIT state is as follows:

If random_priority < station_priority

Then

transmit the packet

Else

defer the transmission to next slot

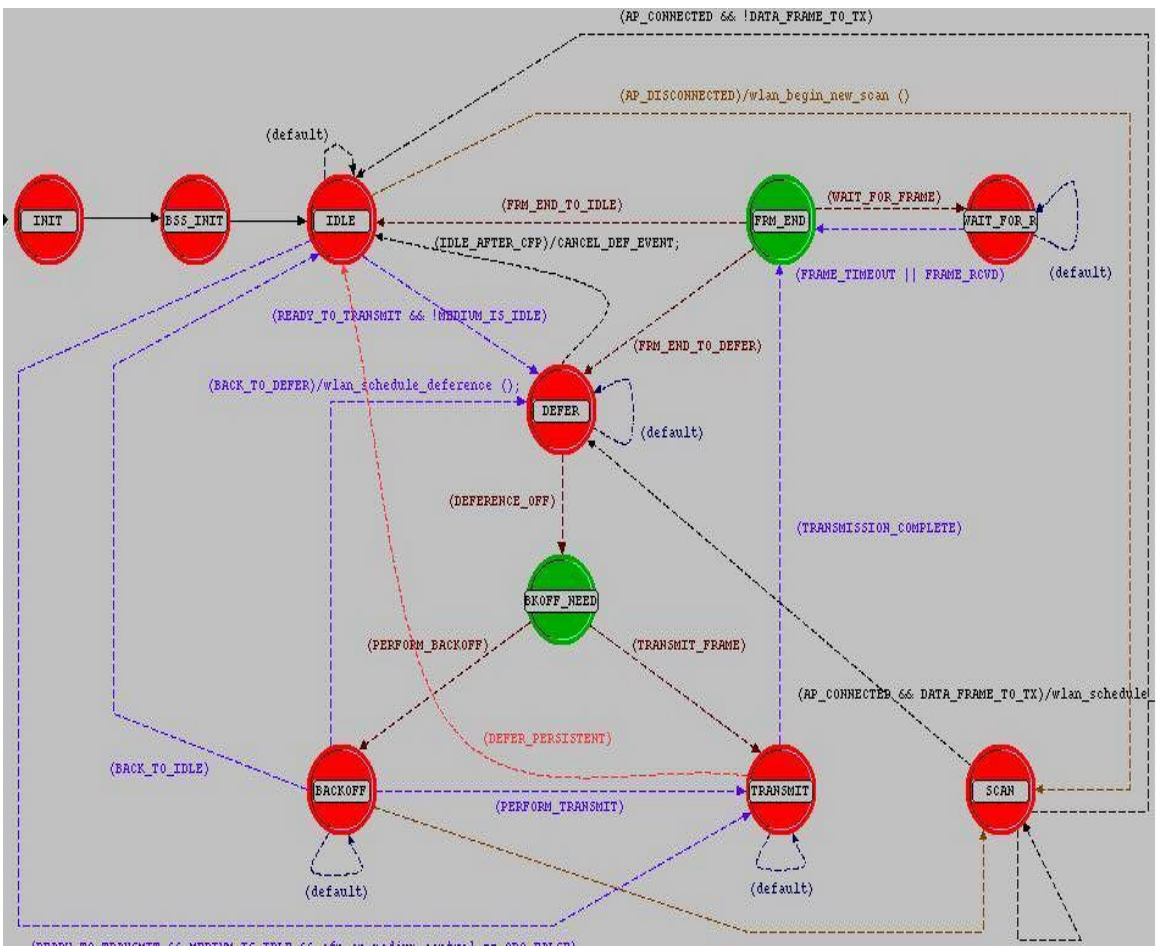

Figure 4: Modified wlan_mac process model for p-Persistent CSMA/CA

6.3 Scenarios and Settings

Two different simulation scenarios were implemented on both the existing contention window scheme and the proposed p-persistent version. The different scenarios are as follows:

i) In the first scenario, the data generation rate was kept constant all the stations and the number of stations (2, 4, 8, 12, 16, and 20) transmitting was varied.

ii) In the second one, the number of stations was made a constant (14) and the data generation rates (0.003, 0.005, 0.007, 0.009 and 0.01) were varied.

In both scenarios, the average throughput and average media access delay of different class priority stations are analyzed. The source stations randomly choose the destination stations. The parameters of the existing contention window scheme and proposed p-persistent version are chosen in such a manner so that the outputs of both the schemes are comparable. In the p-persistent version, low priority class is given a ‘p’ value of 0.3 and high priority class is given a ‘p’ value of 0.5. In the contention window scheme, the low priority class is given additional 800 backoff slots were as high priority is given additional 500 backoff slots.

6.4 Observations and Results

The average throughput and average media access delay of all the transmitting stations are divided into two different class priorities (low and high), which are collected for analysis.

The simulation results (Figures 5 – 8) illustrates that in the first scenario, the proposed p-persistent version the delay is reduced greatly as the load in the network increases while the delay in the contention window scheme increases.

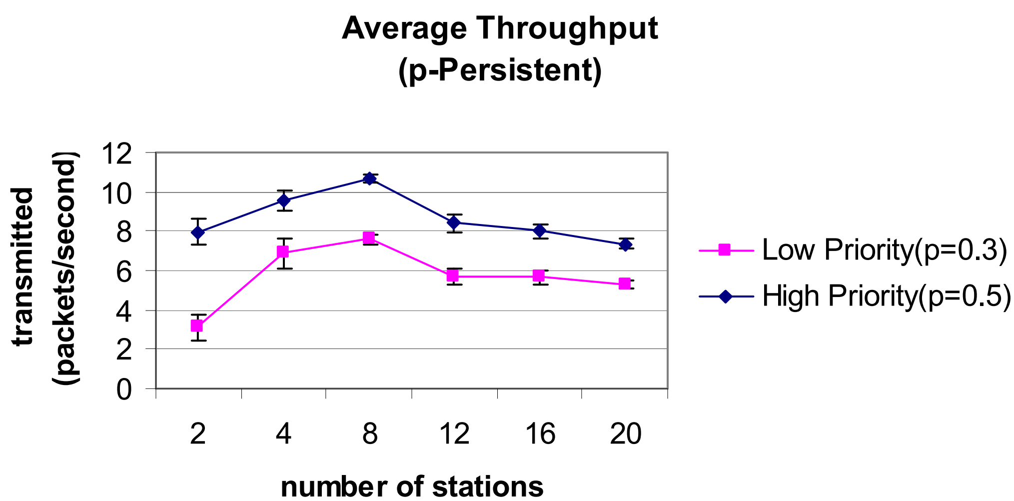

Figure 5: Average Throughput for p-persistent (Scenario 1)

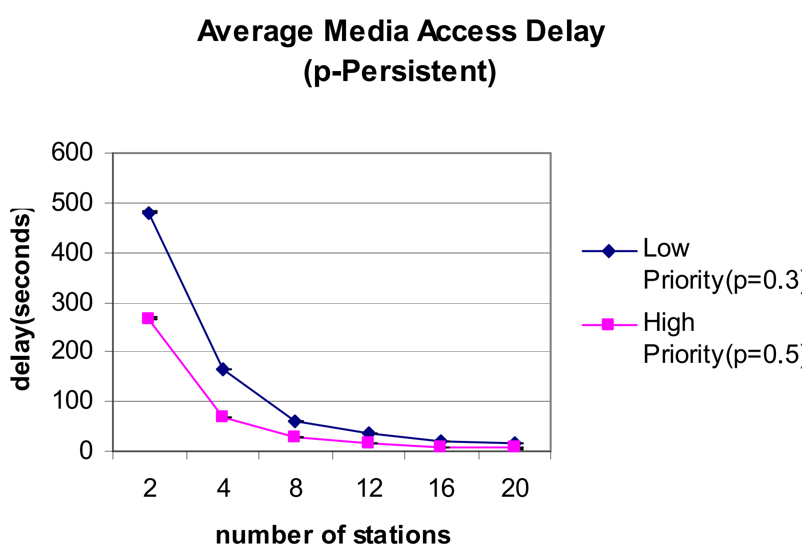

Figure 6: Average Media Access Delay for p-persistent (Scenario 1)

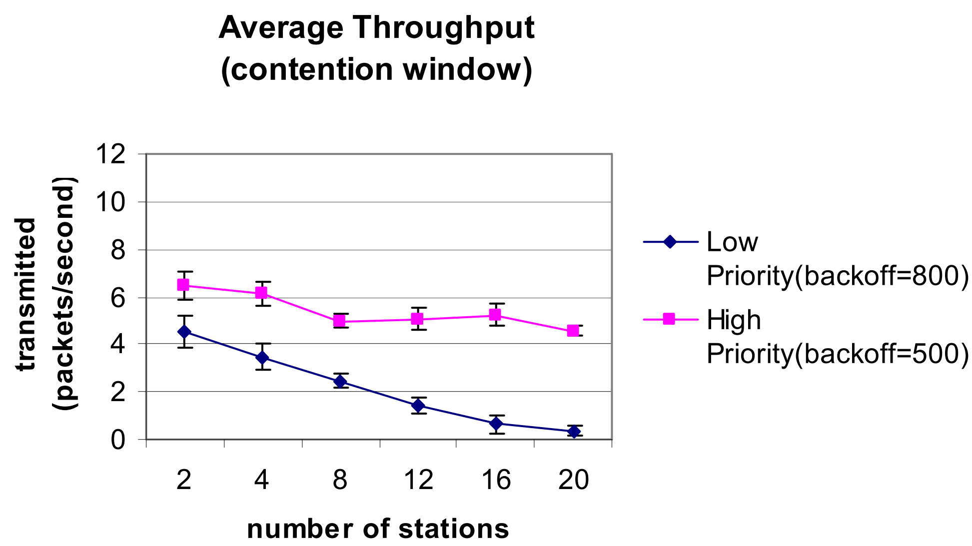

Figure 7: Average Throughput for contention window (Scenario 1)

Moreover, in the first scenario (Figures 5 and 7), throughput of the lower priority class in the contention window scheme drastically decreased compared to that of the proposed p-persistent technique.

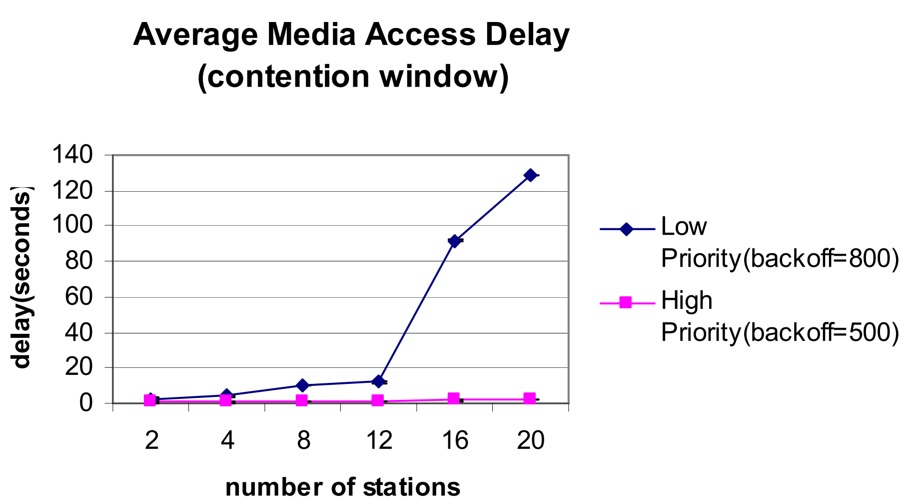

Figure 8: Average Media Access Delay for contention window (Scenario 1)

In the first scenario (Figures 5 and 7), the difference between higher priority and lower priority throughput is a constant in p-persistent version whereas in the contention window scheme it varies.

In the second scenario (Figures 9 – 12), the p-persistent version and contention window scheme almost maintain constant delays were as the throughput decreases when the data generation rate is decreased.

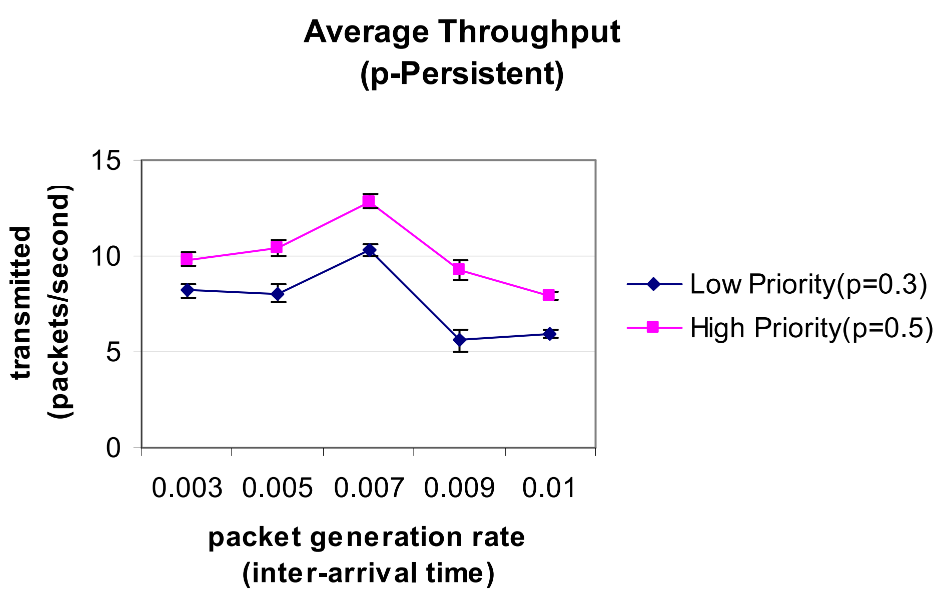

Figure 9: Average Throughput for p-persistent (Scenario 2)

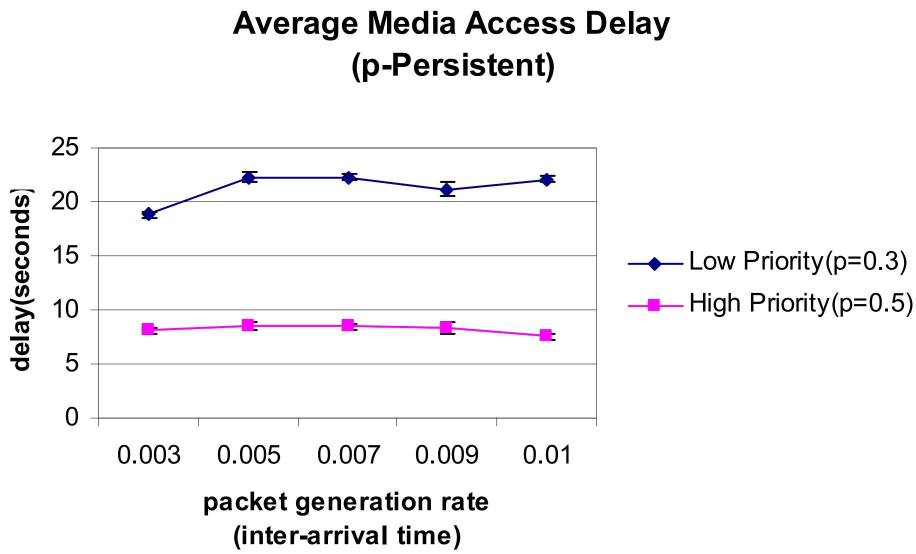

Figure 10: Average Media Access Delay for p-persistent (Scenario 2)

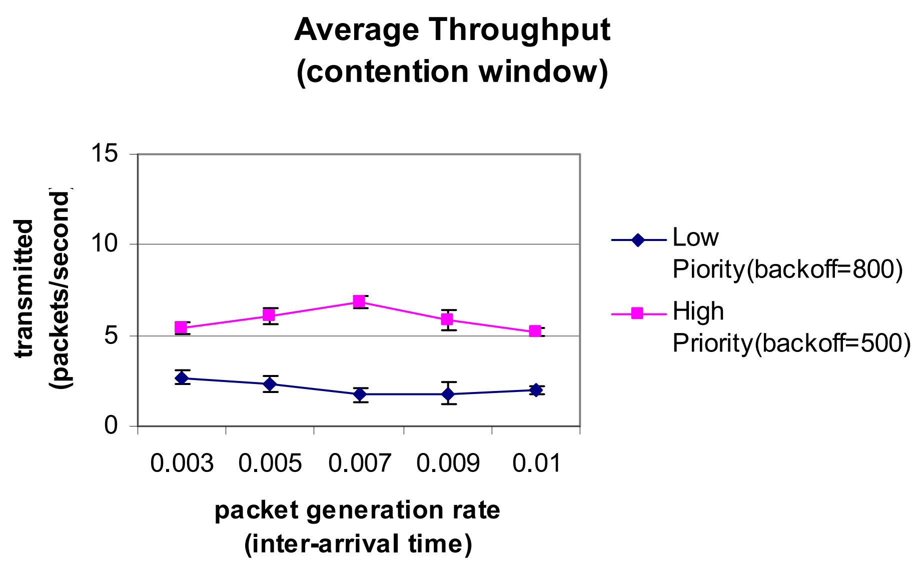

Figure 11: Average Throughput for contention window (Scenario 2)

In the second scenario (Figure 9 and 11), the throughput distinction is a constant in the p-persistent version whereas in the contention window scheme varies.

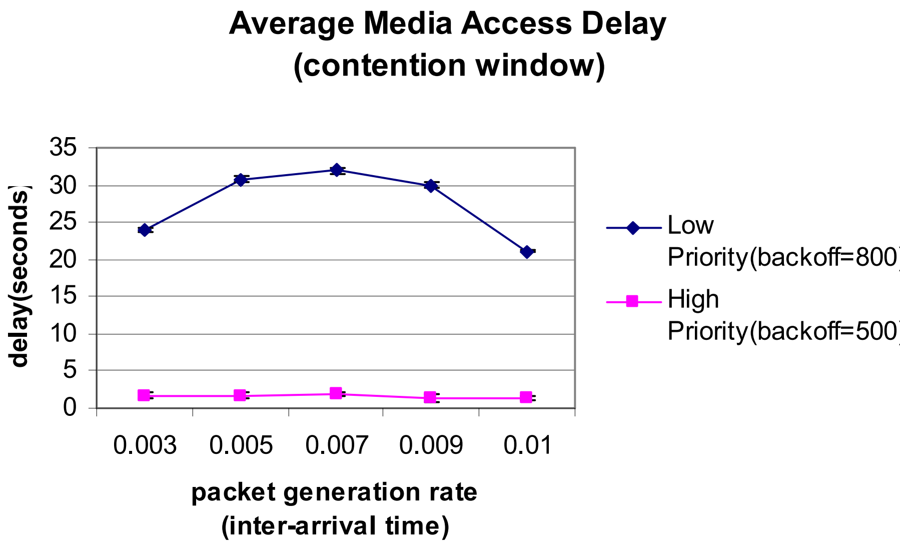

Figure 12: Average Media Access Delay for contention window (Scenario 2)

In the second scenario, the throughput decreases as the data generation rate increased.

Simulation results indicate that the throughput and delay of the existing contention scheme and proposed p-persistent version on both the scenarios are consistent. For example (Figure 8 and 12), illustrates the throughput and delay of 14 station is comparable when the data generation rate is 0.01.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

In the eon of multimedia communication, the design of priority-sensitive network protocols continues to be an important issue, and in particular the broadband wireless links constitute an important class in which prioritization is key to optimizing the overall performance of the network. In this thesis, the proposed p-persistent technique provides service differentiation between two different classes, which can very well be extended to different levels of class-priorities.

The service differentiation using contention window scheme is used as comparison method for the proposed p-persistent version. The two different methods are implemented in wireless LAN environment and their performance (throughput and media access delay) is analyzed. OPNET simulation results indicate that in the p-persistent version, media access delay is reduced as the load in the network increases. Finally, the p-persistent version in the MAC layer can effectively reduce the selection of backoff time when the load increases.

7.2 Future Work

For wireless LANs, well-defined coverage areas simply do not exist. Propagation characteristics are dynamic and unpredictable hence small changes in position or direction may result in dramatic differences in signal strength. If the basic service sets

(BSSs) are not physically very apart, and then two or more BSSs may overlap on the same geographical area.

In this thesis, it had been assumed that the BSSs are far apart so that there or no interference from neighboring BSSs. It would be more appropriate to perform an evaluation study of the proposed MAC scheme by considering the interferences from neighboring BSSs. Another vital avenue for research is to study an automated distributed approach that enables each station to on-line measure parameters needed in calculating the optimal values of ‘p’ and to tune the backoff times accordingly. Since the service differentiation is varied dynamically among the different class-priorities of various stations in the network the throughput can further be optimized.

REFERENCES

[1] Z. Iqbal (2002). Wireless LAN Technology: *Current State and Furture Trends* [Electronic Version]. Retrieved August 23, 2005 from http://www.tml.tkk.fi/Studies/T-110.557/2002/papers/zahed_iqbal.pdf.

[2] IEEE 802.11*,* Part 11: *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications,* Standard, IEEE, Aug. 1999.

[3] IEEE 802.11 WG*,* International Standard [for] Information Technology – Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific Requirements - Part 11:Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, Reference number ISO/IEC 8802-11:1999(E)*, 1999.*

[4] Q. Ni, L. Romdhani, and T. Turletti, *A survey of QoS enhancements for IEEE 802.11 Wireless LAN*, Journal of Wireless and Mobile Computing, John Wiley, Vol. 4, pp. 1-20, 2004, to appear.

[5] P. Ansel, Q. Ni, and T. Turletti, *An efficient scheduling scheme for IEEE 802.11e*. Proc. Of WiOpt (Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks), Cambridge, UK, March 24-26, 2004.

[6] M. Malli, Q. Ni, T. Turletti, and C. Barakat. *Adaptive fair channel allocation for QoS enhancement in IEEE 802.11 wireless LANs.* IEEE ICC 2004 (International Conference on Communications), Paris, June 20-24, 2004.

[7] L. Romdhani, Q. Ni, and T. Turletti, *Adaptive EDCF: enhanced service differentiation for IEEE 802.11 wireless ad hoc networks*, Wireless Communications and Networking Conference, New Orleans, Louisiana, USA, March 16-20, 2003.

[8] Qiang Ni, and Thierry Turletti, *QoS Support for IEEE 802.11 WLAN*, Nova Science Publishers, New York, USA, 2004

[9] L. Kleinrock and F. A. Tobagi. (1975) Packet switching in radio channels: Part 1 - Carrier sense multiple-access modes and their throughput-delay characteristics. IEEE Transactions on Communications, Vol. COM-23, December, pp. 1400 –1416.

[10] F. Cali, M. Conti, and E. Gregori, "*IEEE 802.11Wireless LAN: Capacity Analysis and Protocol Enhancement*," in Proc. IEEE INFOCOM'98, March 1998.

[11] F. Cali, M. Conti, and E. Gregori, "*Dynamic Tuning of the IEEE 802.11 Protocol to Achieve a Theoretical Throughput Limit*," IEEE/ACM Trans. on Networking, Vol. 8, No. 6, pp. 785–799, December 2000.

[12] F. Cali, M. Conti, and E. Gregori, "*IEEE 802.11 Protocol: Design and Performance Evaluation of an Adaptive Backoff Mechanism*," IEEE JSAC, Vol. 18, No. 9, pp. 1774– 1786, September 2000.

[I3] IEEE 802.11ef/D4.0, Draft Supplement to Part II**:** *Wireless Medium Access Control (MAC) and physical layer PHY) specifications: Medium Access Control (MAC) Enhancements for Quality of Service (QoS)*, November 2002.

[14] IEEE 802.1d-1998, Part 3: *Media Access Control (MAC) bridges*, ANSUIEEE Std. 802.1D**,** 1998 edition, 1998.

[15] *Protocol enhancement for IEEE 802.11e EDCF* Xin Wang; Qianli Zhang; Xing Li; Networks, 2004. (ICON 2004). Proceedings, 12th IEEE International Conference on Volume 1, 16-19 Nov. 2004 Page(s): 80 - 84 Vol.1.

[16] OPNET Technologies, Inc, "*Wireless LAN model description,*" http://www.opnet.com/products/library/WLAN_Model_guide1.pdf.

APPENDIX

SOURCE CODE LISTING

The process model implementations of wireless medium access control (MAC) layer are listed for the proposed p-persistent CSMA/CA and existing contention window scheme:

Proposed p-Persistent CSMA/CA:

```
/* OPNET system definitions */
#include <opnet.h>

/* Header Block */

/** Include files **/

#include <math.h>
#include <string.h>
#include "oms_pr.h"
#include "oms_tan.h"
#include "oms_bgutil.h"
#include "wlan_support.h"
#include "oms_auto_addr_support.h"
#include "oms_dist_support.h"
#include "bridge_header.h"
#include "prg_mapping.h"
#include <prg_geo.h>

/** Constants **/

/* Incoming statistics and stream wires.
#define        TRANSMITTER_BUSY_INSTAT              1
#define        LOW_LAYER_INPUT_STREAM        0
#define        LOW_LAYER_OUTPUT_STREAM              0

/* Flags to load different variables based on attribute settings.    */
#define        WLAN_AP                                      1
#define        WLAN_STA                                     0

/* Flags to indicate the medium access mode (PCF/DCF).
#define        PCF_ACTIVE                                   1
#define        PCF_INACTIVE                         0

/* Special value indicating BSS identification is not used.         */
#define        WLAN_BSSID_NOT_USED                  -1

/* Special value indicating radio transceiver frequencies are set    */
/* based on the BSS identification.
#define        WLAN_BSS_BASED_FREQ_USED      -1

/* Special value indicating that the number of back-off slots are    */
```

```
/* not determined yet.                                        */
#define        BACKOFF_SLOTS_UNSET                        -1.0

/* Define a small value (= 1 psec), which will be used to recover    */
/* from double arithmetic precision losts while doing time related   */
/* precision sensitive computations.
#define        PRECISION_RECOVERY                         0.000000000001

/* Special value indicating BSS identification is currently unset.   */
#define        WLANC_BSS_ID_UNKNOWN   -2

/* Define the lowest data transmission rate supported by WLAN MAC.   */
#define        LOWEST_TX_RATE        1000000.0
/* bits/sec */

/* Speed of light (m/s).      */
#define        C      3.0E+08

/* 16 times pi-squared.       */
#define        SIXTEEN_PI_SQ         157.91367

/* Period after which the STA will check for connectivity if scanning
is distance based. */
#define WLANC_CONN_CHK_DIST_INTERVAL  10.0

/* When in the SCAN state, the period that an STA will wait before
trying a new channel.  */
#define WLANC_NEW_SCAN_BEACON_MULT    2.5

/* Physical layer parameters used during roaming/channel scanning.*/
#define WLANC_CHANNEL_COUNT                     11
#define WLANC_FIRST_CHAN_MIN_FREQ               2401.0  /* MHz */
#define WLANC_CHANNEL_BANDWIDTH                 22.0    /* MHz */
#define WLANC_CHANNEL_SPACING                   5.0     /* MHz */
#define WLANC_LAST_CHAN_MIN_FREQ                ((WLANC_CHANNEL_COUNT - 1)
* WLANC_CHANNEL_SPACING + WLANC_FIRST_CHAN_MIN_FREQ)
#define WLANC_CH_STEP_FOR_NO_OVERLAP   ((int) ceil
(WLANC_CHANNEL_BANDWIDTH / WLANC_CHANNEL_SPACING))

/* When virtual scanning is used, two different thresholds are used to
decide  */
/* when the STA must start looking for a new AP, and when a new AP is
/* considered acceptable. This brings in a "hysteresis" which ensures
that the STA does not flip-flop rapidly between APs. */

#define WLANC_ROAM_SCAN_START_VIRTUAL_THRESH  rx_power_threshold
#define WLANC_ROAM_NEW_CONN_VIRTUAL_THRESH
        (rx_power_threshold * 1.1)


/* Define interrupt codes for generating handling interrupts        */
/* indicating changes in deference, frame timeout which infers      */
/* that the collision has occurred, random backoff and transmission  */
/* completion by the physical layer (self interrupts).              */
typedef enum WlanT_Mac_Intrpt_Code
        {
        WlanC_Deference_Off,   /* Deference before frame transmission
```

```
        WlanC_Frame_Timeout,/*No frame rcvd in set duration (infer
                              collision)      */
        WlanC_Backoff_Elapsed,/*Backoff done before frame transmission*/
        WlanC_CW_Elapsed,/* Backoff done after successful frame
                                Transmission   */
        WlanC_Beacon_Tx_Time,  /* Time to transmit beacon frame      */
        WlanC_Cfp_End, /* End of the Contention free period          */
        WlanC_Scan_Timeout,/* End of scan duration for given channel */
        WlanC_AP_Check_Timeout,/* Time to check the connectivity status
                               with the current AP*/
        Deference_Persistent   /* Interrupt code for Deference in
                               p-Persistent CSMA/CA   */

        } WlanT_Mac_Intrpt_Code;

/* Defining codes for the physical layer characteristics type.
typedef enum WlanT_Phy_Char_Code
        {
        WlanC_Frequency_Hopping,
        WlanC_Direct_Sequence,
        WlanC_Infra_Red
        } WlanT_Phy_Char_Code;


/** Global Variables **/

/* Global list of AP position info.
List*   global_ap_pos_info_lptr = OPC_NIL;

/* Global variable to keep note of the nature of the subnet.
/* This variable is initialized to not set.
WlanT_Bss_Identification_Approach bss_id_type = WlanC_Not_Set;


/**     Macro Definitions **/

/** The data frame send flag is set whenever there is a data to be send
    by  the higher layer or the response frame needs to be sent.
     However, in either case the flag will not be set if the receiver is
     busy. Frames cannot be transmitted until medium is idle. Once, the
     medium is available then the station is eligible to transmit
     provided there is a need for backoff. Once the transmission is
     complete then the station will wait for the response provided the
     frame transmitted requires a response (such as RTS and Data
     frames). If response is not needed then the station will defer to
     transmit next packet.
    After receiving a stream interrupt, we need to switch states from
    idle to defer or transmit if there is a frame to transmit and the
    receiver is not busy.If a frame is received indicating that the STA
    should scan, all bets are off, and the STA moves into the scan
    state to look for other APs */

#define READY_TO_TRANSMIT(((intrpt_type == OPC_INTRPT_STRM &&
wlan_flags->data_frame_to_send == OPC_TRUE && (pcf_flag ==
OPC_BOOLINT_DISABLED || (wlan_flags->pcf_active == OPC_FALSE &&
(ap_flag == OPC_BOOLINT_ENABLED || cfp_ap_medium_control ==
OPC_FALSE)))) || fresp_to_send != WlanC_None || wlan_flags->polled ==
```

```
OPC_TRUE || wlan_flags->tx_beacon == OPC_TRUE || wlan_flags->pcf_active
== OPC_TRUE && ap_flag == OPC_BOOLINT_ENABLED)) && !roam_state_ptr-
>scan_mode)

/* When we have a frame to transmit, we move to transmit state if the
   medium was idle for at least a DIFS time, otherwise we go to defer
   state.                                                            */
#define MEDIUM_IS_IDLE                (((current_time - nav_duration +
PRECISION_RECOVERY >= difs_time) && wlan_flags->receiver_busy ==
OPC_FALSE && (current_time - rcv_idle_time + PRECISION_RECOVERY >=
difs_time) && wlan_flags->pcf_active == OPC_FALSE) || wlan_flags-
>forced_bk_end == OPC_TRUE)

/* Change state to Defer from Frm_End, if the input buffers are not
empty or a frame needs to be retransmitted or the station has to respond
to some frame.                                         */
#define FRAME_TO_TRANSMIT             (wlan_flags->data_frame_to_send
== OPC_TRUE || fresp_to_send != WlanC_None || retry_count != 0 ||
wlan_flags->tx_beacon == OPC_TRUE || wlan_flags->cw_required ==
OPC_TRUE)

/* After deferring for either collision avoidance or interframe gap  */
/* the channel will be available for transmission.                   */
#define DEFERENCE_OFF         (intrpt_type == OPC_INTRPT_SELF &&
intrpt_code == WlanC_Deference_Off && wlan_flags->receiver_busy ==
OPC_FALSE)

/*If the Service Differentiation Rule is not satisfied then the
transmission is deferred to the next slot after a deference of 1 ms*/
#define DEFER_PERSISTENT (intrpt_type == OPC_INTRPT_SELF &&
intrpt_code == Deference_Persistent)

/*Issue a transmission complete stat once the packet has successfully*/
/* been transmitted from the source station                          */
#define TRANSMISSION_COMPLETE  (intrpt_type == OPC_INTRPT_STAT &&
op_intrpt_stat () == TRANSMITTER_BUSY_INSTAT)

/* Backoff is performed based on the value of the backoff flag.*/
#define PERFORM_BACKOFF                     (wlan_flags->backoff_flag
== OPC_TRUE || wlan_flags->perform_cw == OPC_TRUE)

/* Need to start transmitting frame once the backoff (self intrpt)   */
/* completed                                                         */
#define BACKOFF_COMPLETED     (intrpt_type == OPC_INTRPT_SELF &&
intrpt_code == WlanC_Backoff_Elapsed && (wlan_flags->receiver_busy ==
OPC_FALSE || wlan_flags->forced_bk_end == OPC_TRUE))

/* Contention Window period, which follows a successful packet       */
/* transmission, is completed.                                       */
#define CW_COMPLETED          (intrpt_type == OPC_INTRPT_SELF &&
intrpt_code == WlanC_CW_Elapsed && (wlan_flags->receiver_busy ==
OPC_FALSE || wlan_flags->forced_bk_end == OPC_TRUE))

/* After transmission the station will wait for a frame response for*/
/* Data and Rts frames.                                             */
#define WAIT_FOR_FRAME          (expected_frame_type != WlanC_None)
```

```
/* Need to retransmit frame if there is a frame timeout and the    */
/* required frame is not received                                 */
#define FRAME_TIMEOUT           (intrpt_type == OPC_INTRPT_SELF &&
intrpt_code == WlanC_Frame_Timeout)

/* If the frame is received appropriate response will be transmitted */
/* provided the medium is considered to be idle                      */
#define FRAME_RCVD             (intrpt_type == OPC_INTRPT_STRM &&
bad_packet_rcvd == OPC_FALSE && i_strm == LOW_LAYER_INPUT_STREAM)

/* Skip backoff if no backoff is needed
#define TRANSMIT_FRAME         (!PERFORM_BACKOFF)

/* Expecting frame response   after data or Rts transmission         */
#define EXPECTING_FRAME        (expected_frame_type != WlanC_None)

/* When the contention window period is over then we go to IDLE state*/
/* if we don't have another frame to send at that moment. If we have */
/* one then we go to TRANSMIT state if we did not sense any activity */
/* on our receiver for a period that is greater than or equal to DIFS*/
/* period; otherwise we go DEFER state to defer and back-off before */
/* transmitting the new frame.
#define BACK_TO_IDLE                  (CW_COMPLETED && wlan_flags-
>data_frame_to_send == OPC_FALSE && !roam_state_ptr->scan_mode)

#define SEND_NEW_FRAME_AFTER_CW       (CW_COMPLETED && wlan_flags-
>data_frame_to_send == OPC_TRUE && MEDIUM_IS_IDLE && !roam_state_ptr-
>scan_mode)

#define DEFER_AFTER_CW                (CW_COMPLETED && wlan_flags-
>data_frame_to_send == OPC_TRUE && !MEDIUM_IS_IDLE && !roam_state_ptr-
>scan_mode)

/* Macros that check the change in the busy status of the receiver.         */
#define RECEIVER_BUSY_HIGH            (intrpt_type == OPC_INTRPT_STAT
&& intrpt_code < TRANSMITTER_BUSY_INSTAT && op_stat_local_read
(intrpt_code) > rx_power_threshold && !wlan_flags->collision)

#define RECEIVER_BUSY_LOW             (intrpt_type == OPC_INTRPT_STAT
&& intrpt_code < TRANSMITTER_BUSY_INSTAT && !wlan_flags->receiver_busy)

#define PERFORM_TRANSMIT              ((BACKOFF_COMPLETED ||
SEND_NEW_FRAME_AFTER_CW))

#define BACK_TO_DEFER                 ((FRAME_RCVD || DEFER_AFTER_CW ||
(wlan_flags->tx_beacon == OPC_TRUE && !wlan_flags->receiver_busy)))

/* Macro to evaluate whether the MAC is in a contention free period. */
#define IN_CFP                                 (pcf_flag ==
OPC_BOOLINT_ENABLED && (cfp_ap_medium_control == OPC_TRUE ||
wlan_flags->pcf_active == OPC_TRUE))

/* After receiving a packet that indicates the end of the current CFP
 go to back to IDLE state if there is no packet to transmit in the CP*/

#define IDLE_AFTER_CFP                (intrpt_type == OPC_INTRPT_STRM&&
!FRAME_TO_TRANSMIT && !IN_CFP)
```

```
/* Macro to cancel the self interrupt for end of deference. It is    */
/* called at the state transition from DEFER to IDLE.                */
#define CANCEL_DEF_EVENT        (op_ev_cancel (deference_evh))

#define FRM_END_TO_IDLE         (!FRAME_TO_TRANSMIT && !EXPECTING_FRAME
&& !IN_CFP)

#define FRM_END_TO_DEFER        (!EXPECTING_FRAME && (FRAME_TO_TRANSMIT
|| IN_CFP))

/* Macros associated with the "SCAN" state. If the scan mode flag is */
/* set, the STA considers itself disconnected from its AP and starts */
/* scanning for a new AP-- only in DCF STAs.                         */
#define AP_DISCONNECTED         (roam_state_ptr->scan_mode == OPC_TRUE)

#define AP_CONNECTED            (roam_state_ptr->scan_mode == OPC_FALSE)

#define DATA_FRAME_TO_TX        (wlan_flags->data_frame_to_send ==
OPC_TRUE)

#define SCAN_TIMEOUT            (intrpt_type == OPC_INTRPT_SELF &&
intrpt_code == WlanC_Scan_Timeout)

#define SCAN_AFTER_CW           (CW_COMPLETED && AP_DISCONNECTED)

/* End of Header Block */

/* State variable definitions */
typedef struct
        {
        /* Internal state tracking for FSM */
        FSM_SYS_STATE
        /* State Variables */
        int                                     retry_count;
        int                                     intrpt_type;
        WlanT_Mac_Intrpt_Code                   intrpt_code;
        int                                     my_address;
        Objid                                   my_objid;
        Objid                                   my_node_objid;
        Objid                                   my_subnet_objid;
        Objid                                   tx_objid;
        Objid                                   txch_objid;
        Objid                                   rx_objid;
        Objid                                   rxch_objid;
        OmsT_Pr_Handle                          own_process_record_handle;
        List*                                   hld_list_ptr;
        double                                  operational_speed;
        int                                     frag_threshold;
        int                                     packet_seq_number;
        int                                     packet_frag_number;
        int                                     destination_addr;
        Sbhandle                                fragmentation_buffer_ptr;
        Sbhandle                                common_rsmbuf_ptr;
        WlanT_Mac_Frame_Type                    fresp_to_send;
        double                                  nav_duration;
        int                                     rts_threshold;
```

```
int                                       duplicate_entry;
WlanT_Mac_Frame_Type                      expected_frame_type;
int                                       remote_sta_addr;
double                                    backoff_slots;
Stathandle                                packet_load_handle;
double                                    intrpt_time;
Packet *
wlan_transmit_frame_copy_ptr;
Stathandle                                backoff_slots_handle;
int                                       instrm_from_mac_if;
int                                       outstrm_to_mac_if;
int                                       num_fragments;
OpT_Packet_Size                           remainder_size;
List*                                     defragmentation_list_ptr;
WlanT_Mac_Flags*                          wlan_flags;
OmsT_Aa_Address_Handle                    oms_aa_handle;
double                                    current_time;
double                                    rcv_idle_time;
Pmohandle                                 hld_pmh;
int                                       max_backoff;
char                                      current_state_name [32];
Stathandle                                hl_packets_rcvd;
Stathandle                                media_access_delay;
Stathandle                                ete_delay_handle;
Stathandle                                global_ete_delay_handle;
Stathandle                                global_throughput_handle;
Stathandle                                global_load_handle;
Stathandle                        global_dropped_data_handle;
Stathandle                                global_mac_delay_handle;
Stathandle                        ctrl_traffic_rcvd_handle_inbits;
Stathandle                        ctrl_traffic_sent_handle_inbits;
Stathandle                                ctrl_traffic_rcvd_handle;
Stathandle                                ctrl_traffic_sent_handle;
Stathandle                        data_traffic_rcvd_handle_inbits;
Stathandle                        data_traffic_sent_handle_inbits;
Stathandle                                data_traffic_rcvd_handle;
Stathandle                                data_traffic_sent_handle;
double                                    sifs_time;
double                                    slot_time;
int                                       cw_min;
int                                       cw_max;
double                                    difs_time;
double                                    plcp_overhead_control;
double                                    plcp_overhead_data;
Stathandle                                channel_reserv_handle;
Stathandle                                retrans_handle;
Stathandle                                throughput_handle;
int                                       long_retry_limit;
int                                       short_retry_limit;
int                                       retry_limit;
WlanT_Mac_Frame_Type                      last_frametx_type;
Evhandle                                  deference_evh;
Evhandle                                  backoff_elapsed_evh;
Evhandle                                  frame_timeout_evh;
double                                    eifs_time;
int                                       i_strm;
Boolean                                   wlan_trace_active;
```

```
SimT_Pk_Id                              pkt_in_service;
Stathandle                              bits_load_handle;
int                                     ap_flag;
Boolean                                 bss_flag;
int                                     ap_mac_address;
int                                     hld_max_size;
double                                  max_receive_lifetime;
int                                     accept_large_packets;
WlanT_Phy_Char_Code                     phy_char_flag;
OpT_Packet_Size                         total_hlpk_size;
Stathandle                              drop_packet_handle;
Stathandle                              drop_packet_handle_inbits;
Log_Handle                              drop_pkt_log_handle;
Log_Handle                              config_log_handle;
int                                     drop_pkt_entry_log_flag;
int                                     packet_size;
double                                  receive_time;
Ici*                                    llc_iciptr;
double                                  rx_power_threshold;
int                                     bss_id;
int                                     pcf_retry_count;
int                                     poll_fail_count;
int                                     max_poll_fails;
List*                                   cfpd_list_ptr;
int                                     pcf_queue_offset;
double                                  beacon_int;
Sbhandle                                pcf_frag_buffer_ptr;
Packet *                        wlan_pcf_transmit_frame_copy_ptr;
int                                     pcf_num_fragments;
OpT_Packet_Size                         pcf_remainder_size;
int*                                    polling_list;
int                                     poll_list_size;
int                                     poll_index;
double                                  pifs_time;
Evhandle                                beacon_evh;
Evhandle                                cfp_end_evh;
SimT_Pk_Id                              pcf_pkt_in_service;
int                                     pcf_flag;
Boolean                                 active_pc;
int                                     cfp_prd;
int                                     cfp_offset;
double                                  cfp_length;
Boolean                                 ap_relay;
OpT_Packet_Size                         total_cfpd_size;
OpT_Packet_Size                         packet_size_dcf;
OpT_Packet_Size                         packet_size_pcf;
double                                  receive_time_dcf;
double                                  receive_time_pcf;
Boolean                                 cfp_ap_medium_control;
int                                     pcf_network;
int                                     beacon_eff_mode;
int                                     channel_num;
int                                     eval_bss_id;
WlanT_Roam_State_Info*                  roam_state_ptr;
WlanT_Rx_State_Info*                    rx_state_info_ptr;
double                          ap_connectivity_check_interval;
double                          ap_connectivity_check_time;
```

```
        Evhandle                      ap_connectivity_check_evhndl;
        WlanT_AP_Position_Info*               conn_ap_pos_info_ptr;
        WlanT_Sta_Mapping_Info*               my_sta_info_ptr;
        WlanT_Bss_Mapping_Info*               my_bss_info_ptr;
        PrgT_Mutex*                           mapping_info_mutex;
        double                                my_priority;
        int                                   ack_seq_num;
        int                                   dat_seq_num;
        } wlan_mac_p-Persistent_state;

/** state (TRANSMIT) enter executives **/
FSM_STATE_ENTER_UNFORCED (4, "TRANSMIT", state4_enter_exec,
"wlan_mac_sample1 [TRANSMIT enter execs]")
FSM_PROFILE_SECTION_IN (wlan_mac_sample1 [TRANSMIT enter execs],
state4_enter_exec)
{
/** In this state following intrpts can occur:               **/
/** 1. Data arrival from application layer.                  **/
/** 2. Frame (DATA,ACK,RTS,CTS) rcvd from PHY layer.         **/
/** 3. Busy intrpt stating that frame is being rcvd.         **/
/** 4. Collision intrpt means more than one frame is rcvd.   **/
/** 5. Transmission completed intrpt from physical layer     **/
/** Queue the packet for Data Arrival from the higher layer, **/
/** and do not change state.                                 **/
/** After Transmission is completed change state to FRM_END  **/
/** No response is generated for any lower layer packet arrival**/
/* Prepare transmission frame by setting appropriate  */
/* fields in the control/data frame.                  */
/* Skip this routine if any frame is received from the       */
/* higher or lower layer(s)                                   */
printf("\nAddress(TxENT) :%d\tPKT_SEQ_NUM :
%d",my_address,packet_seq_number);
if (wlan_flags->immediate_xmt == OPC_TRUE)
{
/* Initialize the contention window size for the      */
/* packets that are sent without backoff for the      */
/* first time, if in case they are retransmitted.     */
        max_backoff = cw_min;

        tmp_pty=op_dist_uniform(1);
        printf("\nTesting PTY1 : %.1f",tmp_pty);
        if(tmp_pty < my_priority)
        {
             printf("\nTransmit1");
             wlan_frame_transmit ();
        }
        else
        {
             printf("\nDeference1");
            deference_evh = p_intrpt_schedule_self(op_sim_time()+0.001,
            Deference_Persistent);
        }
}
else
{
        printf("\nDestination Address : %d",destination_addr);
        wlan_frame_transmit ();
```

```
}
/* Start the transmission.                    */
/* Reset the immediate transmission flag.     */
wlan_flags->immediate_xmt = OPC_FALSE;
}

else if (wlan_flags->rcvd_bad_packet == OPC_FALSE && intrpt_type ==
OPC_INTRPT_SELF)
{
/* If it is a PCF enabled MAC then make sure that     */
/* the interrupt was not PCF related. Start the       */
/* transmission, if the delivered self interrupt is   */
/* an interrupt that was just brought us into this    */
/* state.                                             */
if ((pcf_flag == OPC_BOOLINT_DISABLED || intrpt_code ==
WlanC_Deference_Off || intrpt_code == WlanC_Backoff_Elapsed ||
intrpt_code == WlanC_CW_Elapsed) && !(intrpt_code ==
WlanC_Beacon_Tx_Time || intrpt_code == WlanC_AP_Check_Timeout))
{
        wlan_frame_transmit ();
        tmp_pty=op_dist_uniform(1);
        printf("\nTesting PTY2 : %.1f",tmp_pty);
        if(tmp_pty < my_priority)
        {
           printf("\nTransmit2");
           wlan_frame_transmit ();
        }
        else
        {
           printf("\nDeference2");
           deference_evh = op_intrpt_schedule_self (op_sim_time()+0.001,
                          Deference_Persistent);
        }
/* Check whether the forced transmission (end */
/* of backoff) flag is set.                   */
if (wlan_flags->forced_bk_end == OPC_TRUE)
{
/* Reset the flag.                            */
wlan_flags->forced_bk_end = OPC_FALSE;
/* This flag indicates a rare case: at the    */
/* exact time when we completed our backoff   */
/* and started our transmission, we also      */
/* started receiving a packet. Hence, mark    */
/* the currently being received packet as a   */
/* bad packet.                                */
wlan_flags->rcvd_bad_packet = OPC_TRUE;
}
if (wlan_trace_active)
{
/* Determine the current state name.                             */
strcpy (current_state_name, "transmit");
}

/* Unlock the mutex that serializes accessing the               */
/* roaming related information of this MAC.                     */
op_prg_mt_mutex_unlock (roam_state_ptr->roam_info_mutex);
}
```

Existing Contention Window Scheme:

```
/* OPNET system definitions */
#include <opnet.h>

/* Header Block */

/** Include files **/

#include <math.h>
#include <string.h>
#include "oms_pr.h"
#include "oms_tan.h"
#include "oms_bgutil.h"
#include "wlan_support.h"
#include "oms_auto_addr_support.h"
#include "oms_dist_support.h"
#include "bridge_header.h"
#include "prg_mapping.h"
#include <prg_geo.h>

/** Global Variables **/

/* Global list of AP position info.                          */
List*   global_ap_pos_info_lptr = OPC_NIL;
/* Global variable to keep note of the nature of the subnet. */
/* This variable is initialized to not set.                  */
WlanT_Bss_Identification_Approach bss_id_type = WlanC_Not_Set;

/* State variable definitions */
typedef struct
        {
        /* Internal state tracking for FSM */
        FSM_SYS_STATE
        /* State Variables */
        int                                     retry_count;
        int                                     intrpt_type;
        WlanT_Mac_Intrpt_Code                   intrpt_code;
        int                                     my_address;
        Objid                                   my_objid;
        Objid                                   my_node_objid;
        Objid                                   my_subnet_objid;
        Objid                                   tx_objid;
        Objid                                   txch_objid;
        Objid                                   rx_objid;
        Objid                                   rxch_objid;
        OmsT_Pr_Handle                          own_process_record_handle;
        List*                                   hld_list_ptr;
        double                                  operational_speed;
        int                                     frag_threshold;
        int                                     packet_seq_number;
        int                                     packet_frag_number;
        int                                     destination_addr;
        Sbhandle                                fragmentation_buffer_ptr;
        Sbhandle                                common_rsmbuf_ptr;
```

```
WlanT_Mac_Frame_Type            fresp_to_send;
double                          nav_duration;
int                             rts_threshold;
int                             duplicate_entry;
WlanT_Mac_Frame_Type            expected_frame_type;
int                             remote_sta_addr;
double                          backoff_slots;
Stathandle                      packet_load_handle;
double                          intrpt_time;
Packet *
wlan_transmit_frame_copy_ptr;
Stathandle                      backoff_slots_handle;
int                             instrm_from_mac_if;
int                             outstrm_to_mac_if;
int                             num_fragments;
OpT_Packet_Size                 remainder_size;
List*                           defragmentation_list_ptr;
WlanT_Mac_Flags*                wlan_flags;
OmsT_Aa_Address_Handle          oms_aa_handle;
double                          current_time;
double                          rcv_idle_time;
Pmohandle                       hld_pmh;
int                             max_backoff;
char                            current_state_name [32];
Stathandle                      hl_packets_rcvd;
Stathandle                      media_access_delay;
Stathandle                      ete_delay_handle;
Stathandle                      global_ete_delay_handle;
Stathandle                      global_throughput_handle;
Stathandle                      global_load_handle;
Stathandle
global_dropped_data_handle;
Stathandle                      global_mac_delay_handle;
Stathandle
ctrl_traffic_rcvd_handle_inbits;
Stathandle
ctrl_traffic_sent_handle_inbits;
Stathandle                      ctrl_traffic_rcvd_handle;
Stathandle                      ctrl_traffic_sent_handle;
Stathandle
data_traffic_rcvd_handle_inbits;
Stathandle
data_traffic_sent_handle_inbits;
Stathandle                      data_traffic_rcvd_handle;
Stathandle                      data_traffic_sent_handle;
double                          sifs_time;
double                          slot_time;
int                             cw_min;
int                             cw_max;
double                          difs_time;
double                          plcp_overhead_control;
double                          plcp_overhead_data;
Stathandle                      channel_reserv_handle;
Stathandle                      retrans_handle;
Stathandle                      throughput_handle;
int                             long_retry_limit;
int                             short_retry_limit;
```

```
int                                     retry_limit;
WlanT_Mac_Frame_Type                    last_frametx_type;
Evhandle                                deference_evh;
Evhandle                                backoff_elapsed_evh;
Evhandle                                frame_timeout_evh;
double                                  eifs_time;
int                                     i_strm;
Boolean                                 wlan_trace_active;
SimT_Pk_Id                              pkt_in_service;
Stathandle                              bits_load_handle;
int                                     ap_flag;
Boolean                                 bss_flag;
int                                     ap_mac_address;
int                                     hld_max_size;
double                                  max_receive_lifetime;
int                                     accept_large_packets;
WlanT_Phy_Char_Code                     phy_char_flag;
OpT_Packet_Size                         total_hlpk_size;
Stathandle                              drop_packet_handle;
Stathandle                              drop_packet_handle_inbits;
Log_Handle                              drop_pkt_log_handle;
Log_Handle                              config_log_handle;
int                                     drop_pkt_entry_log_flag;
int                                     packet_size;
double                                  receive_time;
Ici*                                    llc_iciptr;
double                                  rx_power_threshold;
int                                     bss_id;
int                                     pcf_retry_count;
int                                     poll_fail_count;
int                                     max_poll_fails;
List*                                   cfpd_list_ptr;
int                                     pcf_queue_offset;
double                                  beacon_int;
Sbhandle                                pcf_frag_buffer_ptr;
Packet *                 wlan_pcf_transmit_frame_copy_ptr;
int                                     pcf_num_fragments;
OpT_Packet_Size                         pcf_remainder_size;
int*                                    polling_list;
int                                     poll_list_size;
int                                     poll_index;
double                                  pifs_time;
Evhandle                                beacon_evh;
Evhandle                                cfp_end_evh;
SimT_Pk_Id                              pcf_pkt_in_service;
int                                     pcf_flag;
Boolean                                 active_pc;
int                                     cfp_prd;
int                                     cfp_offset;
double                                  cfp_length;
Boolean                                 ap_relay;
OpT_Packet_Size                         total_cfpd_size;
OpT_Packet_Size                         packet_size_dcf;
OpT_Packet_Size                         packet_size_pcf;
double                                  receive_time_dcf;
double                                  receive_time_pcf;
Boolean                                 cfp_ap_medium_control;
```

```
	int					pcf_network;
	int					beacon_eff_mode;
	int					channel_num;
	int					eval_bss_id;
	WlanT_Roam_State_Info*			roam_state_ptr;
	WlanT_Rx_State_Info*			rx_state_info_ptr;
	double			ap_connectivity_check_interval;
	double			ap_connectivity_check_time;
	Evhandle			ap_connectivity_check_evhndl;
	WlanT_AP_Position_Info*			conn_ap_pos_info_ptr;
	WlanT_Sta_Mapping_Info*				my_sta_info_ptr;
	WlanT_Bss_Mapping_Info*				my_bss_info_ptr;
	PrgT_Mutex*				mapping_info_mutex;
	double					my_priority;
	int					ack_seq_num;
	int					dat_seq_num;
	int					bk1;
	int					bk2;
	int					bkr;
	double					tsum1;
	double					tsum2;
	} wlan_mac_ContentionWindowScheme_state;

static void wlan_mac_sv_init ()
	{
	Objid
	mac_params_comp_attr_objid;
	Objid					params_attr_objid;
	Objid				pcf_params_comp_attr_objid;
	Objid					subpcf_params_attr_objid;
	Objid					chann_objid;
	int					num_chann;
	double					tx_power;
	int					i;
	Objid					statwire_objid;
	int					num_statwires;
	double					threshold;
	void*					temp_ptr;
	char					mutex_name_str [64];
	int					roaming_cap_flag;

	/** 1. Initialize state variables.				**/
	/** 2. Read model attribute values in variables.	**/
	/** 3. Create global lists					**/
	/** 4. Register statistics handlers			**/
	FIN (wlan_mac_sv_init ());

	/* object id of the surrounding processor.		*/
	my_objid = op_id_self ();

	/* Obtain the node's object identifier			*/
	my_node_objid = op_topo_parent (my_objid);

	/* Obtain subnet objid.					*/
	my_subnet_objid = op_topo_parent (my_node_objid);

	/* Obtain the values assigned to the various attributes	*/
```

```
op_ima_obj_attr_get (my_objid, "Wireless LAN Parameters",
&mac_params_comp_attr_objid);
params_attr_objid = op_topo_child (mac_params_comp_attr_objid,
OPC_OBJTYPE_GENERIC, 0);

/* Determine the assigned MAC address.                              */
op_ima_obj_attr_get (my_objid, "Address", &my_address);

/* Obtain an address handle for resolving WLAN MAC addresses.
oms_aa_handle = oms_aa_address_handle_get ("MAC Addresses",
"Address");

/* Obtain the BSS_Id attribute to determine if BSS based network
is used */
op_ima_obj_attr_get (params_attr_objid, "BSS Identifier",
&bss_id);

/* Register the log handles and related flags.
config_log_handle      = op_prg_log_handle_create
(OpC_Log_Category_Configuration, "Wireless Lan", "MAC
Configuration", 128);
drop_pkt_log_handle    = op_prg_log_handle_create
(OpC_Log_Category_Protocol,      "Wireless Lan", "Data packet
Drop",  128);
drop_pkt_entry_log_flag = 0;

/* Update the global variable if this is the first node to come
up. If not the first node, then check for mismatches. A subnet
can be a traditional subnet (i.e. a subnet with one BSS, this is
the existing model) or a BSS based subnet where for every node
the attribute BSS_Id is set to indicate to which BSS a node
belongs. If the global is set to traditional subnet and the this
node has its BSS_Id attribute set then log a warning message
and recover by considering the BSS_Id attribute setting as not
used. If the global is set to BSS based subnet and this node is
not using its BSS_Id attribute then log an error message and
stop the simulation.                                         */
if (bss_id_type == WlanC_Not_Set)
        {
        if (bss_id ==  WLAN_BSSID_NOT_USED)
                {
                bss_id_type = WlanC_Entire_Subnet ;
                }
        else
                {
                bss_id_type = WlanC_Bss_Divided_Subnet ;
                }
        }

/* Configuration error checking */
if (bss_id_type == WlanC_Entire_Subnet && bss_id
!=WLAN_BSSID_NOT_USED)
{
/*Recoverable mismatch, log warning and continue by enforcing */
/*traditional subnet,i.e.force the bss_id variable to not used*/
/* Write the warning message.                                 */
op_prg_log_entry_write (config_log_handle,"WARNING:\n"
```

```
" A node with an explicit BSS \n"
" assignment was found in a pure \n"
" subnet.\n"
"ACTION:\n"
" The BSS identifier is set to\n"
" the default value.\n"
"CAUSE:\n"
" There are some nodes in the\n"
" network which have their BSS\n"
" identifiers set to the default\n"
" while the others have the\n"
" default setting.\n"
"SUGGESTION:\n"
" Ensure that all nodes have the\n"
" BSS identifier set to the default\n"
" value or all of them are explicitly\n"
" assigned.\n");
        }
else if (bss_id_type == WlanC_Bss_Divided_Subnet && bss_id ==
WLAN_BSSID_NOT_USED)
{
/* Unrecoverable error-- not all BSS IDs have been configured.
Suppose in the wlan what the BSS ID should be, hence terminate*/
wlan_mac_error ("BSS ID not set in a node which belongs to a
network in which some BSS IDs are set","Please set a non-default
BSS ID on all nodes in the network", OPC_NIL);
}

/* Use the subnet ID as the BSS ID if it is set to "NOT USED".
if (bss_id_type == WlanC_Entire_Subnet)
{
        bss_id = my_subnet_objid;

/* Add the BSS ID into the BSS ID list, which is later going to
be used while selecting channels for BSSs.*/
wlan_bss_id_list_manage (bss_id, "add");
}

/* Get model attributes.      */
op_ima_obj_attr_get (params_attr_objid, "Data Rate",
&operational_speed);
op_ima_obj_attr_get (params_attr_objid, "Fragmentation
Threshold", &frag_threshold);
op_ima_obj_attr_get (params_attr_objid, "Rts Threshold",
&rts_threshold);
op_ima_obj_attr_get (params_attr_objid, "Short Retry Limit",
&short_retry_limit);
op_ima_obj_attr_get (params_attr_objid, "Long Retry Limit",
&long_retry_limit);
op_ima_obj_attr_get (params_attr_objid, "Access Point
Functionality", &ap_flag);
op_ima_obj_attr_get (params_attr_objid, "Buffer Size",
&hld_max_size);
op_ima_obj_attr_get (params_attr_objid, "Max Receive Lifetime",
&max_receive_lifetime);
op_ima_obj_attr_get (params_attr_objid, "Large Packet
Processing", &accept_large_packets);
```

```
op_ima_obj_attr_get_dbl (my_node_objid, "Priority",
&my_priority);

/* Get simulation attributes. */
op_ima_sim_attr_get (OPC_IMA_TOGGLE, "WLAN Beacon Efficiency
Mode", &beacon_eff_mode);

/* Initialize the retry limit for the current frame to long
retry limit.   */
retry_limit = long_retry_limit;

/* Extract beacon and PCF parameters.                        */
op_ima_obj_attr_get (params_attr_objid, "PCF Parameters",
&pcf_params_comp_attr_objid);
subpcf_params_attr_objid = op_topo_child
(pcf_params_comp_attr_objid, OPC_OBJTYPE_GENERIC, 0);
op_ima_obj_attr_get (subpcf_params_attr_objid, "PCF
Functionality", &pcf_flag);
op_ima_obj_attr_get (subpcf_params_attr_objid, "CFP Beacon
Multiple", &cfp_prd);
op_ima_obj_attr_get (subpcf_params_attr_objid, "CFP Offset",
&cfp_offset);
op_ima_obj_attr_get (subpcf_params_attr_objid, "CFP Interval",
&cfp_length);
op_ima_obj_attr_get (subpcf_params_attr_objid, "Max Failed
Polls", &max_poll_fails);
op_ima_obj_attr_get (subpcf_params_attr_objid, "Beacon
Interval", &beacon_int);

ap_relay = OPC_TRUE;

/* Check if there is an active AP controlling the medium during
the CFP.*/
if ((ap_flag == OPC_BOOLINT_ENABLED) && (pcf_flag ==
OPC_BOOLINT_ENABLED))
        active_pc= OPC_TRUE;
else
        active_pc= OPC_FALSE;

/* Load the appropriate physical layer characteristics.
op_ima_obj_attr_get (params_attr_objid, "Physical
Characteristics", &phy_char_flag);

/* Obtain the receiver valid packet power threshold value used
   by the statwires from the receiver into the MAC module.   */
op_ima_obj_attr_get (params_attr_objid, "Packet Reception-Power
 Threshold", &rx_power_threshold);
op_ima_obj_attr_get (params_attr_objid, "Transmit Power",
&tx_power);

/* Based on physical characteristics settings set appropriate
values to the variables.                                 */
switch (phy_char_flag)
        {
        case WlanC_Frequency_Hopping:
                {
                /* Slot duration in terms of seconds.         */
```

```
            slot_time = 50E-06;

            /* Short interframe gap in terms of seconds.  */
            sifs_time = 28E-06;

            /* PLCP overheads, which include the preamble and
               header, in terms of seconds.               */
            plcp_overhead_control = 128E-06;
            plcp_overhead_data    = 128E-06;

            /* Minimum contention window size for selecting
            backoff slots. */
            cw_min = 15;
            /* Maximum contention window size for selecting
            backoff slots. */
            cw_max = 1023;
            break;
            }

        case WlanC_Direct_Sequence:
            {
            /* Slot duration in terms of seconds.               */
            slot_time = 20E-06;

            /* Short interframe gap in terms of seconds.        */
            sifs_time = 10E-06;

            /* PLCP overheads, which include the preamble and
            header, in terms of seconds.                        */
            plcp_overhead_control = 192E-06;
            plcp_overhead_data    = 192E-06;

            /* Minimum contention window size for selecting
            backoff slots. */
            cw_min = 31;

            /* Maximum contention window size for selecting
            backoff slots. */
            cw_max = 1023;

            if(my_priority == 0.3) //LOW PRIORITY CLASS
            {
                    cw_min = 500;
                    cw_max = 1023;
            }
            else if(my_priority == 0.8)//HIGH PRIORITY CLASS
            {
                cw_min = 100;
                cw_max = 500;
            }
            Else //NORMAL PACKETS CLASS FOR ACKs
            {
                    cw_min = 31;
                    cw_max = 100;
            }

            break;
```

```
        }

case WlanC_Infra_Red:
        {
        /* Slot duration in terms of seconds.
        slot_time = 8E-06;

        /* Short interframe gap in terms of seconds.
        sifs_time = 7E-06;

        /* PLCP overheads, which include the preamble and
        header, in     */
        /* terms of seconds. Infra-red supports
        transmission of parts  */
        /* of the PLCP header at the regular data
        transmission rate,     */
        /* which can be higher than mandatory lowest data
        rate.                  */
        plcp_overhead_control = 57E-06;
        plcp_overhead_data    = 25E-06 + (ceil
        (32000000.0 / operational_speed) / 1E6);

        /* Minimum contention window size for selecting
        backoff slots. */
        cw_min = 63;

        /* Maximum contention window size for selecting
        backoff slots. */
        cw_max = 1023;
        break;
        }

default:
        {
        wlan_mac_error ("Unexpected Physical Layer
        Characteristic encountered.", OPC_NIL, OPC_NIL);
        break;
        }
}

/** By default stations are configured for IBSS unless
an Access Point is found,**/
/** then the network will have an infrastructure BSS
configuration.                        **/
bss_flag = OPC_FALSE;

/* Computing DIFS interval which is interframe gap
between successive frame transmissions.       */
difs_time = sifs_time + 2 * slot_time;

/* If the receiver detects that the received frame is
erroneous then it will set the network allocation vector
to EIFS duration. */
eifs_time = difs_time + sifs_time + WLAN_ACK_DURATION +
plcp_overhead_control;
```

```
		/** PIFS duration is used by the AP operating under PCF
		to gain priority to access the medium **/
		pifs_time = sifs_time + slot_time;

		/* Creating list to store data arrived from higher
		layer.  */
		hld_list_ptr = op_prg_list_create ();

		/* If the station is an AP, and PCF supported, create
		separate PCF queue list for higher layer. */
		if ((ap_flag == OPC_BOOLINT_ENABLED) && (pcf_flag ==
		OPC_BOOLINT_ENABLED))
			cfpd_list_ptr = op_prg_list_create ();
		else
			cfpd_list_ptr = OPC_NIL;

		/* Initialize segmentation and reassembly buffers.   */
		defragmentation_list_ptr = op_prg_list_create ();
		fragmentation_buffer_ptr = op_sar_buf_create
		(OPC_SAR_BUF_TYPE_SEGMENT,    OPC_SAR_BUF_OPT_PK_BNDRY);
		common_rsmbuf_ptr        = op_sar_buf_create
		(OPC_SAR_BUF_TYPE_REASSEMBLY, OPC_SAR_BUF_OPT_DEFAULT);


	/* Create the mutex that will be used to serialize calling of
	prg_mapping functions, which read/write global model related
	mapping information, under multi-threaded execution with
	multiple CPUs. */
	mapping_info_mutex = op_prg_mt_mutex_create
	(OPC_MT_MUTEX_READER_WRITER, 0, "WLAN Mapping Info Mutex");

	FOUT;
	}
/** state (BKOFF_NEEDED) enter executives **/
FSM_STATE_ENTER_FORCED (3, "BKOFF_NEEDED", state3_enter_exec,
"wlan_mac_sample2 [BKOFF_NEEDED enter execs]")
FSM_PROFILE_SECTION_IN (wlan_mac_sample2 [BKOFF_NEEDED enter execs],
state3_enter_exec)
{
/** In this state we determine whether a back-off is necessary for the
    frame we are trying to transmit. It is needed when station
    preparing to transmit frame discovers that the medium is busy or
    the station is responding to the frame. Following a successful
    packet transmission, again a back-off procedure is performed for a
    contention window period as stated in 802.11 standard.   **/
/** If backoff needed then check whether the station completed its **/
/** backoff in the last attempt. If not then resume the backoff      **/
/** from the same point, otherwise generate a new random number      **/
/** for the number of backoff slots.                                 **/

/* Checking whether backoff is needed or not.                        */
if (wlan_flags->backoff_flag == OPC_TRUE || wlan_flags->perform_cw ==
OPC_TRUE)
{
if (backoff_slots == BACKOFF_SLOTS_UNSET)
{
/* Compute backoff interval using binary exponential process.*/
```

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

/* The number of possible slots grows exponentially until it */
/* exceeds a fixed limit.*/
if (max_backoff > cw_max)
{
        max_backoff = cw_max;
}
/* Obtain a uniformly distributed random integer between 0 and       */
/* the minimum contention window size. Scale the number of           */
/* slots according to the number of retransmissions.                 */
backoff_slots = floor (op_dist_uniform (max_backoff + 1));
}
if(my_priority==0.3)
{
        tsum1=tsum1+ backoff_slots ;
        bk1++;
}
else if(my_priority==0.8)
{
        tsum2=tsum2+ backoff_slots;
        bk2++;
}
else
        bkr++;
printf("\nBACKOFF SLOTS: %.1f FOR STATION: %d BackOff STATE Count:%d
backoffSLOTS:%.1f",backoff_slots,my_address,((my_priority==0.1)?(bk1):(
bk2)),((my_priority==0.1)?(tsum1):(tsum2)));

/* Set a timer for the end of the backoff interval.   */
if(my_priority==0.3)
{
backoff_slots = backoff_slots + 800;
intrpt_time = (current_time + backoff_slots * slot_time);
printf("\tIntrpt Time(PTY(%.1f)) : %f Station ID :
%d",my_priority,intrpt_time,my_address);
}
else if(my_priority==0.8)
{
        backoff_slots = backoff_slots +500;
        intrpt_time = (current_time + backoff_slots * slot_time);
        printf("\tIntrpt Time(PTY(%.1f)) : %f Station Id :
        %d",my_priority,intrpt_time,my_address);
}
else
{
```

```
intrpt_time = (current_time + backoff_slots * slot_time);
printf("\tIntrpt Time(Rx) : %f Station ID :%d",intrpt_time,my_address);
}
printf("\nINTRPT TIME SET: %f Station ID : %d",intrpt_time,my_address);

/* Scheduling self interrupt for backoff.                              */
if (wlan_flags->perform_cw == OPC_TRUE)
        backoff_elapsed_evh = op_intrpt_schedule_self (intrpt_time,
        WlanC_CW_Elapsed);
else
        backoff_elapsed_evh = op_intrpt_schedule_self (intrpt_time,
        WlanC_Backoff_Elapsed);
/* Reporting number of backoff slots as a statistic.                   */
op_stat_write (backoff_slots_handle, backoff_slots);
}

}
FSM_PROFILE_SECTION_OUT (wlan_mac_contentwindowscheme [BKOFF_NEEDED
enter execs], state3_enter_exec)

/** End of state (BKOFF_NEEDED) Enter executives **/
```

VITA

Vijay Gurusamy

Candidate for the Degree of

Master of Science

Thesis: SERIVCE DIFFERENTIATION USING p-PERSISTENT CSMA/CA

Major Field: Computer Science

Biographical:

Personal Data: Born in Trichy, Tamil Nadu, India on October 26, 1980, the son of Mr. R. Gurusamy and Mrs. Saroja Gurusamy.

Education: Received Bachelor of Engineering in Computer Science and Engineering from Bharathidasan University, Trichy, India in May 2002. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in December, 2005.

Professional Experience: August 2004 – December 2005 worked as Software Programmer for Soil Water Forage Analytical Labs (SWFAL), Dr. Halin Zhang, Plant and Soil Sciences Department, Oklahoma State University, Stillwater.

Name: Vijay Gurusamy Date of Degree: December 2005

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: SERVICE DIFFERENTIATION USING p-PERSISTENT CSMA/CA

Pages of Study: 52 Candidate for the Degree of Master of Science

Major Field: Computer Science

Scope and Method of Study: The problem of service differentiation in wireless LANs is of utmost importance compared to wired networks. Multimedia communication via wireless LANs needs larger bandwidth compared to non-multimedia communications. Hence, a service differentiation scheme to enhance the distinction of service offered to different class priorities is essential. The objective of this thesis is to study the problem of providing service differentiation using proposed p-Persistent CSMA approach and the existing contention window scheme. The areas of this study include service differentiation schemes in p-Persistent CSMAs, various contention window schemes and a number of adaptive service differentiation schemes. Also, a service differentiation rule is used to distinguish the services offered to various classes of traffic.

Findings and Conclusions: Service differentiation for different classes of priorities using proposed p-Persistent version and existing contention window scheme were studied. The performance (average throughput & average media access delay) of both p-Persistent CSMA and contention window scheme was studied in depth. A service differentiation rule was implemented to distinguish the priorities among different classes of traffic. The parameters of both the schemes were chosen to be comparable. The p-Persistent CSMA version and contention window scheme works successfully for the two different scenarios (Scenario 1: load varied & data generation rate constant, Scenario 2: data generation rate varied & load constant). Finally, the p-Persistent version when analyzed offered service differentiation with lower delay as the network load increases.

ADVISOR'S APPROVAL: Dr. Venkatesh Sarangan