ENSC 835-3: NETWORK PROTOCOLS AND PERFORMANCE
CMPT 885-3: HIGH-PERFORMANCE NETWORKS

Performance Evaluation and Enhancement of Wireless Local Area Networks

PROJECT REPORT

(Spring 2002)

Jiaqing (James) Song

E-mail: songs@acm.org

Website: www.songssoft.com/james/courses/cmpt885

TABLE OF CONTENTS

ABSTRACT

Unlike the huge bandwidth can be achieved by the wired networks, the bandwidth of wireless network is much more limited due to the "air", the physical median used to transfer signal by WLAN, is not only error prone but also very "expensive" (the license fee for using a commercial frequency band is billions of dollars), so improving the performance of the WLAN is a very important topic to research.

In this project, IEEE802.11 standard and the corresponding OPNET Nodes/Process models are studied in detail. Then a survey of the current research works on improving WLAN performance is done. And then several important methods for improving the performance of the WLAN, from the tuning-up of Physical Layer Related Parameters and IEEE802.11 Parameters to the tuning-up of Link Layer Protocol (Media Access Control) are implemented and simulated with OPNET. Some work on the tuning-up of Network Layer Protocol (Transfer Control Protocol) is also done as an additional and challenging part of my project. Finally, some problems discovered in this project are discussed and future works are suggested.

Moreover, future works suggested in last year's project "Wireless Ethernet Performance"[18] (includes: "Analysis of the PHY layer that the WLAN models in OPNET work do not allow; Disable the models of certain layers (TCP/IP) as they affect network performance too; Can tune the back off selection algorithm based on network conditions; Investigate of effect of different data traffic") are accomplished in this project. Part of the work (Packet Error Generator) in last year's project "Performance of TCP Protocol Running over WLAN 802.11 with the Snoop Protocol"[17] is implemented with enhanced features.

OPNET 8.0.c is chosen as the simulation tool in this project.

INTRODUCTION

1.1 Introduction: IEEE 802.11 WLAN

The IEEE802.11 standard defines the protocol and compatible interconnection of data communication equipment via the "air", radio or infrared, in a local area network (LAN). It includes both the PHY layer and the MAC layer among the ISO 7-layer network model. [19]

In the MAC sub-layer, a fundamental access method of Distributed Coordination Function (DCF) and an optional method of Point Coordination Function (PCF) are used. The DCF is also known as Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA). It is an asynchronous access method based on the contention for the usage of shared channel. The PCF proves a contention-free access mechanism through RTS/CTS exchange.

The IEEE802.11 protocol also includes authentication, association, and re-association services, an optional encryption/decryption procedure, power management to reduce power consumption in mobile stations, and a point coordination function for time-bounded transfer of data.

But there are some problems with the Wireless LAN. First, the media of WLAN is error prone such that the bits error rate (BER) of WLAN is very high compared to the wired and networks. Second, Carrier Sensing is difficult in the wireless networks because of the incapability of a station to listen to their own transmissions to detect a collision. Third, Hidden Terminal problem can also reduce the performance of wireless networks.

1.2 Introduction: OPNET WLAN models

WLAN station:
The wireless station node model represents an IEEE802.11 wireless LAN station. The node model consists of an ON/OFF (active/inactive) traffic source, a sink, a Wireless LAN interface and Wireless LAN receiver/transmitter.

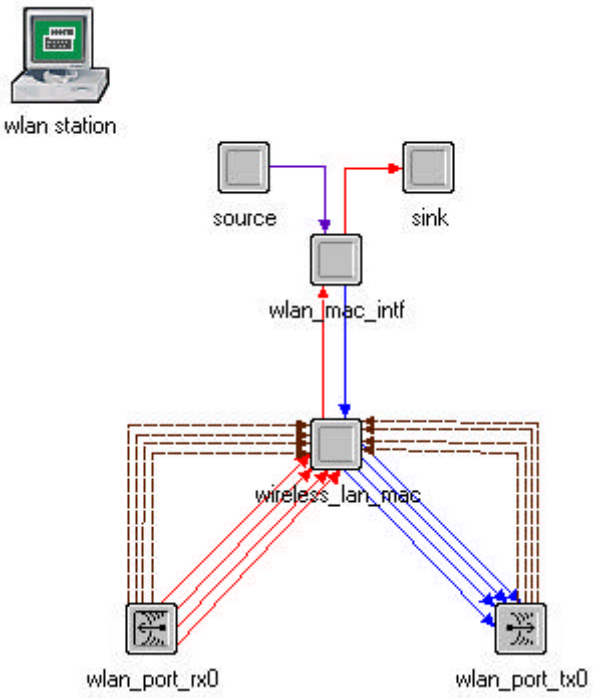

Figure 1.1: OPNET node model - WLAN Station (wlan_station)

WLAN workstation & server:

The WLAN workstation node model represents a workstation with client-server applications running over TCP/IP and UDP/IP. The workstation supports one underlying WLAN connection at 1 Mbps, 2 Mbps, 5.5 Mbps, and 11 Mbps.

The WLAN server node model represents a server with server applications running over TCP/IP and UDP/IP. This node supports one underlying IEEE 802.11 connection at 1 Mbps, 2 Mbps, 5.5 Mbps, and 11 Mbps. The operational speed is determined by the connected link's data rate.

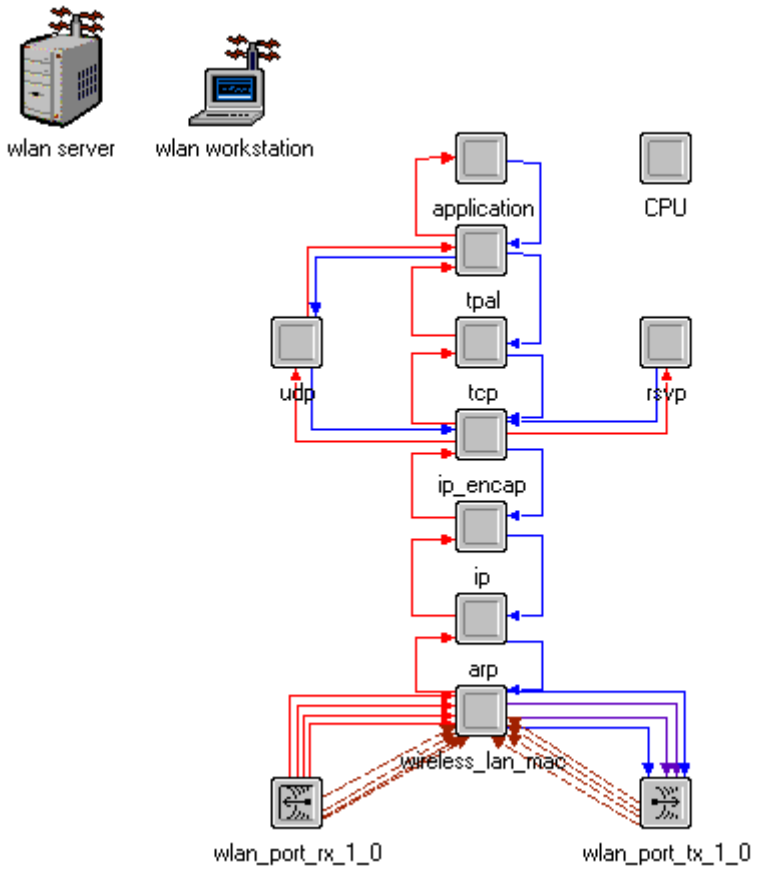

Figure 1.2: OPNET node model - WLAN Workstation/Server (wlan_wkstn/wlan_server)

WLAN Access Point (Wireless Router):

The WLAN access point node model represents a wireless LAN based router with Ethernet interface. It can be used as a router in wireless networks or can connect the wireless network to the wired networks.

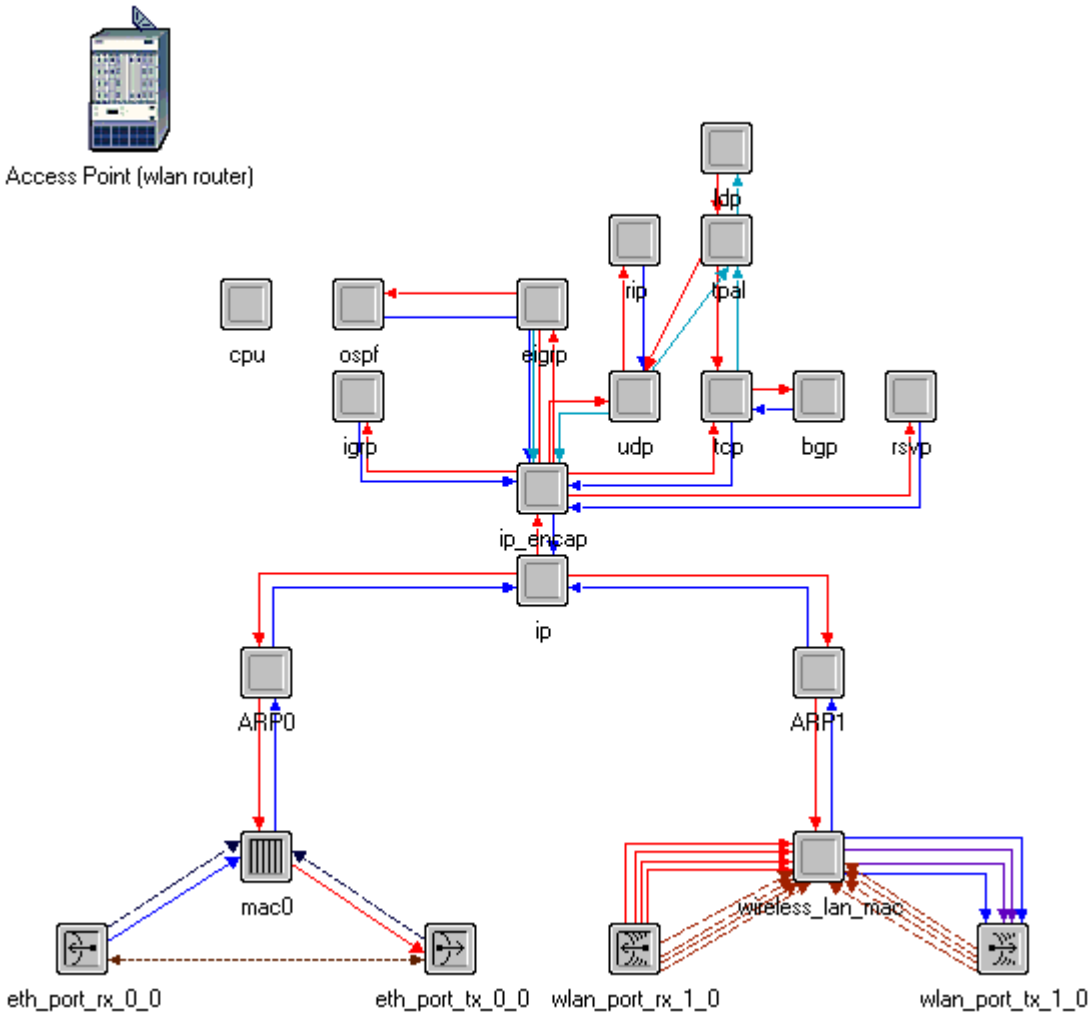

Figure 1.3: OPNET node model - WLAN Access Point (wlan_ethernet_router)

Independent Basic Service Set (IBSS):

IEEE 802.11 WLAN is built up from BSSs. A BSS is a set of stations that communicate with one another. The dash line with arrows in the Figure 1.4 is a demonstration of the communication routes between stations. When all the stations in the BSS can communicate directly with each other and there is no connection to a wired network, the BSS is called an independent BSS or an adhoc network.

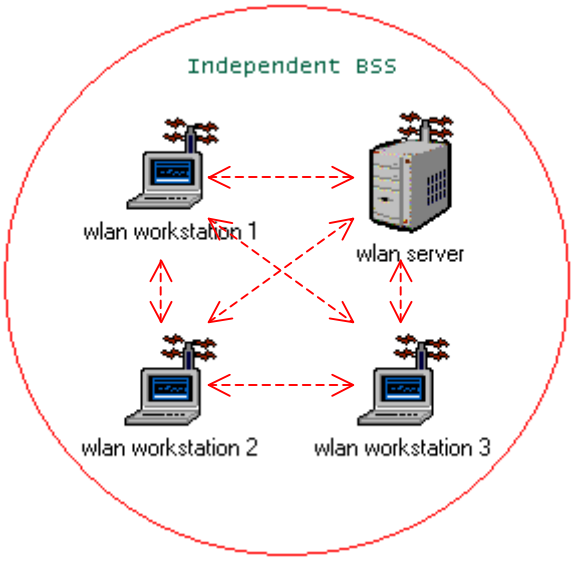

Figure 1.4: OPNET scenario - WLAN Independent BSS

Infrastructure Basic Service Set (IBSS):
When an Access Point (Wireless Router) is added to the Independent BSS, the station in the BSS cannot communication with each directly. All the communications are relayed by the access point and the IBSS is called an infrastructure BSS now. The dash line with arrows in the Figure 1.5 is a demonstration of the communication routes between stations.

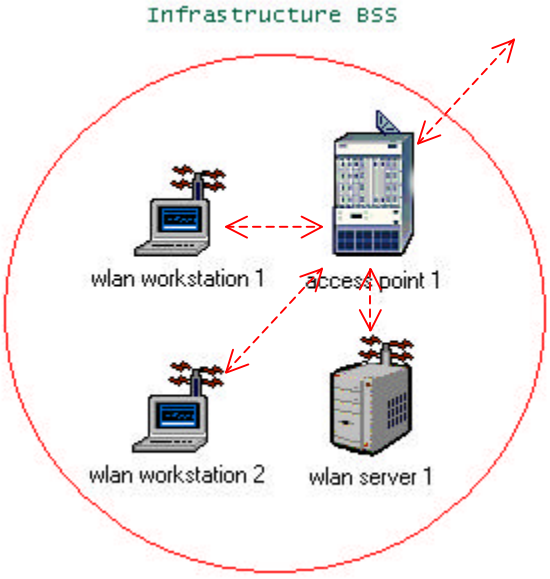

Figure: OPNET scenario - WLAN Infrastructure BSS

A typical scenario of Mixed LAN & WLAN Networks:

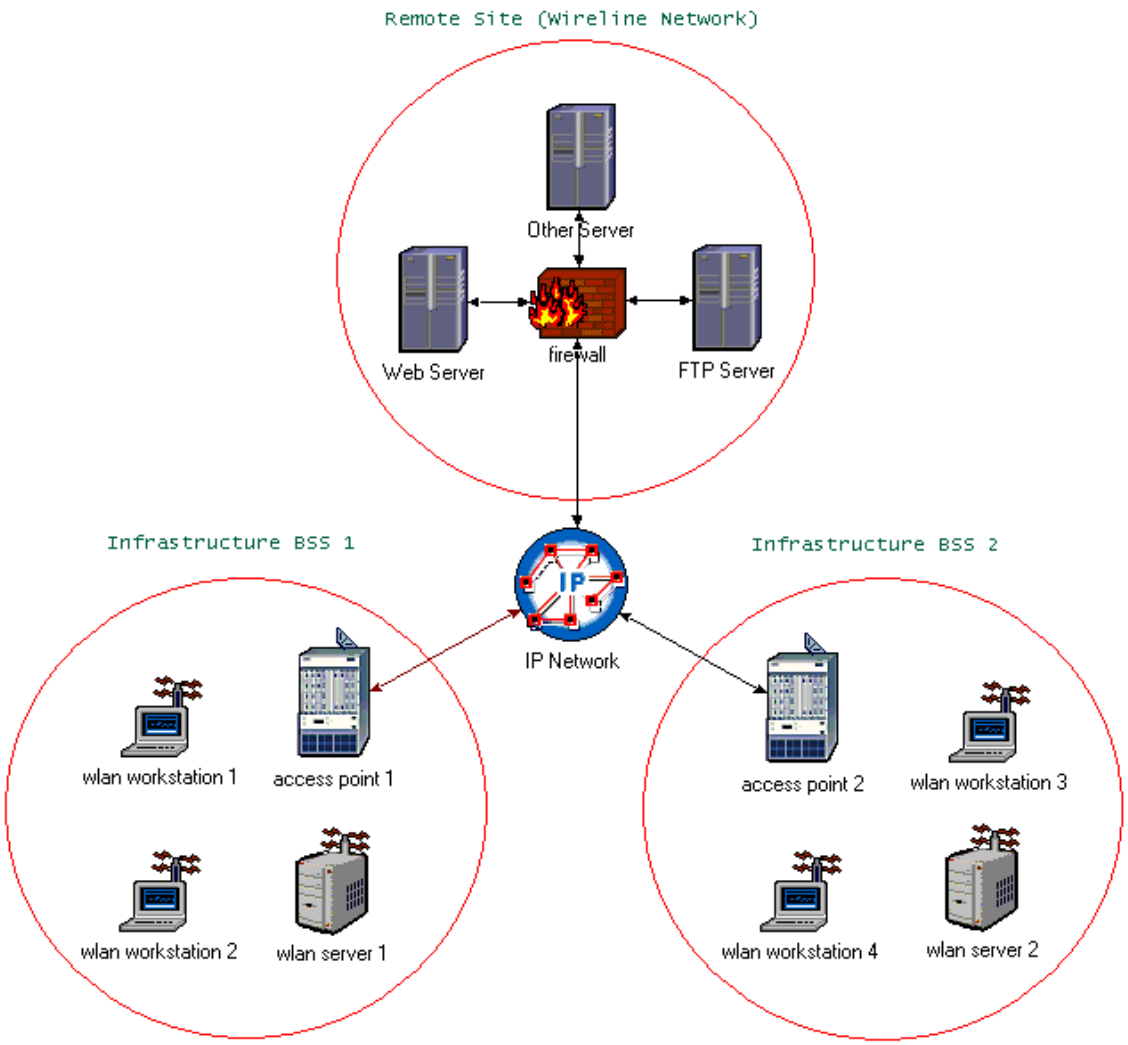

Figure: OPNET scenario - Mixed LAN & WLAN Networks

For more information about the IEEE802.11 and OPNET, please refer to [19][20] in the References Section.

1.3 Introduction: Performance Enhancement of WLAN

According to tens of research papers published recently, the methods for improving the performance of wireless LAN can be categorized into the following categories:

a) Use enhanced hardware on the Physical Layer to achieve optimized PHY layer related parameters (such as shorter Slot Time and shorter SIFS), so as to improve the wlan performance.
b) Tune up the WLAN parameters (such as proper Fragmentation Threshold and RTS Threshold) to improve the wlan performance.
c) Substitute the standard back-off algorithm with an adaptive back-off algorithm on MAC layer

to improve the wlan performance.
d) Use proxy approach on the link-layer to improve the wlan performance (such as snoop, SMART snoop protocol)
e) Use reliable link-layer approach to improve the wlan performance (AIRMAIL)
f) Use split-connection approach to improve the wlan performance (I-TCP, M-TCP)

In this project, methods a) b) and c) are successfully implemented and simulated.
Method d) is studied and simulated as a challenging and tentative part of the project.

PERFORMANCE TUNE-UP: PHY CHARACTERISTIC RELATED PARAMETERS

In this part of the project, I will simulate and study the effect of PHY characteristic related parameters. There are three kinds of PHY characteristics pre-defined by OPNET model. They are "Frequency Hopping", "Direct Sequence" and "Infra Red". But OPNET does not provide any customized PHY parameters.

3.1 Implementation

To study the effect of PHY characteristic related parameters, I modify the OPNET wlan_mac process model to add "Slot Time", "SIFS Time", "Minimum Contention Window" and "Maximum Contention Window" parameters into the Wireless LAN Parameters Table. These four parameters will take effect if the "Customized" item is selected as "Physical Characteristics" (Figure 3.1). If the "Physical Characteristics" is set to any other values, the customized parameters will be disabled automatically.

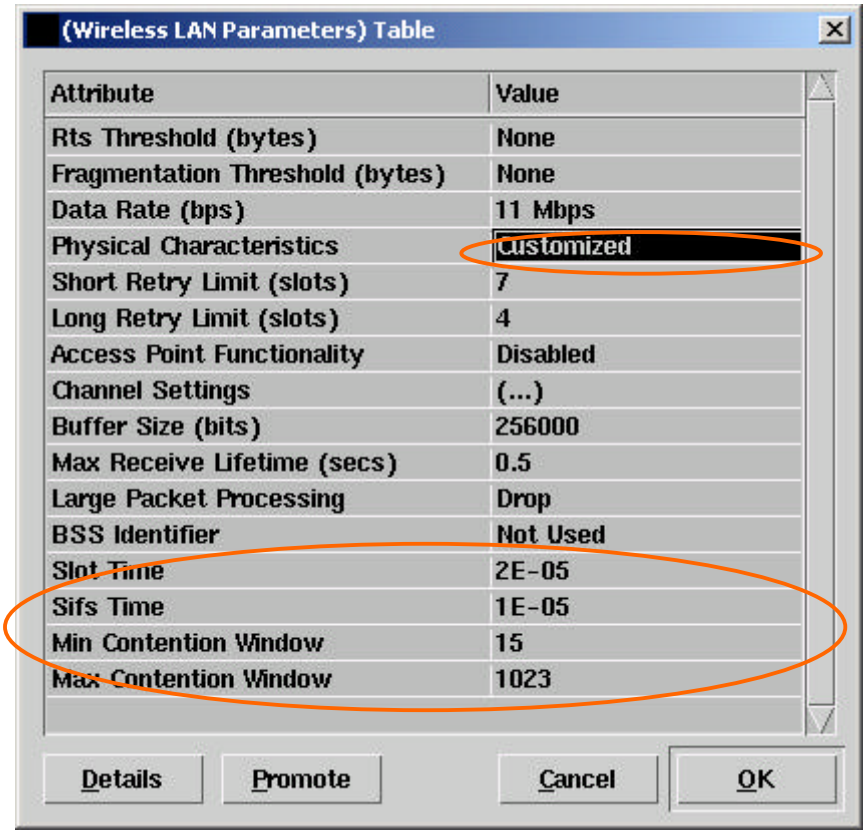

Figure 3.1: the customized physical lay parameters

3.2 Scenario and Settings

In the simulation, a simple scenario with two wlan stations is used (Figure 3.2). The two stations send data to each other at an average rate of about 820Kbits/s during the simulation. Traffic type is listed in Table 3.1 in detail.

The wlan_station (Figure 1.1), instead of wlan_workstation (Figure 1.2), is chosen to do the simulation because there is no interference from the TCP and Higher Layers in wlan_station node model, so it is suitable for studying the protocols on the MAC layer.

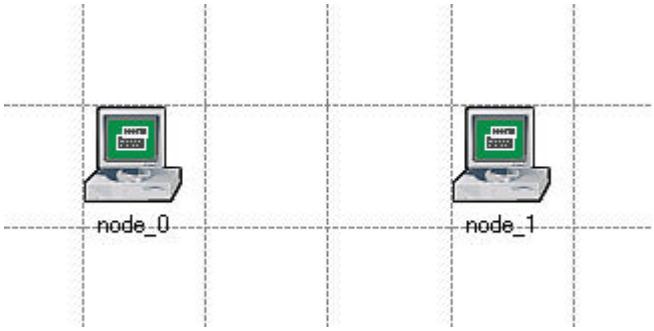

Figure 3.2 OPNET Scenario with two wlan stations

Attribute	Value
Time start to send packets (second)	constant (2)
Time stop sending packets (second)	Never
Packet interarrival time (second)	exponential (0.01)
Packet size (bytes)	exponential (1024)
Segmentation size (bytes)	No segmentation

Table 3.1 Traffic generation parameters

3.3 Simulation, Results and Analysis

Simulation on "Slot time & SIFS"

The first pair of simulation is done to study the effect of different "Slot time & Short Inter-Frame Space (SIFS)" to the WLAN performance. Parameters for the "simulation set 1" and "simulation set 2" are listed in the Table 3.2.

	Simulation set 1	Simulation set 2
PHY characteristic	Frequency hopping	Customized
Slot time (s)	5.0E-05	2.0E-05
SIFS (s)	2.8E-05	1.0E-05
Min contention window size	15	15
Max contention window size	1023	1023
WLAN bandwidth (bps)	11M	11M
WLAN buffer size (bits)	256k	256k

Table 3.2 Parameters of two sets of simulation on the "Slot time, SIFS"

In the simulation, *Media Access Delay* of node_0 is collected for analysis.

Media Access Delay*: The total of queue and contention delays of data packets received by WLAN MAC from higher layer. For each packet, the delay is recorded when the packet is sent to the physical layer for the first time.*

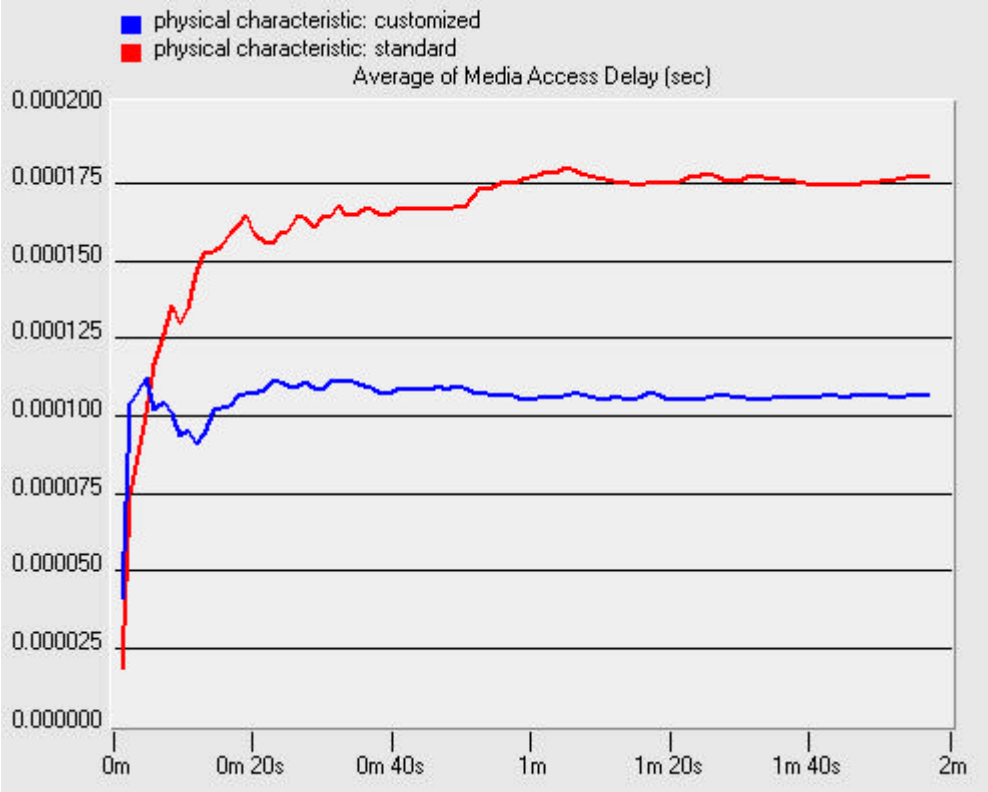

Figure 3.3: Simulation results of different Slot Time & SIFS

From Figure 3.3, it is very obvious that a smaller "Slot time" and "SIFS" value can achieve a shorter Average Media Access Delay, so as to improve the performance of the wireless network.

However, because "Slot time" and "SIFS" are direct reflection of the hardware characteristics of the physical layer and they are tightly related to which kind of hardware we are choosing, so this part is not the emphasis of the project.

Simulation on "Min Contention Window"

The second pair of simulation is done to study the effect of different "Min Contention Window" to the WLAN performance. Parameters for the "simulation set 3" and "simulation set 4" are listed in the Table 3.3.

	Simulation set 3	**Simulation set 4**
PHY characteristic	Customized	Customized
Slot time (s)	5.0E-05	5.0E-05
SIFS (s)	2.8E-05	2.8E-05
Min contention window size	7	63
Max contention window size	1023	1023
WLAN bandwidth (bps)	11M	11M

WLAN buffer size (bits)	256k	256k

Table 3.3 Parameters of two sets of simulation on the "Min Contention Window"

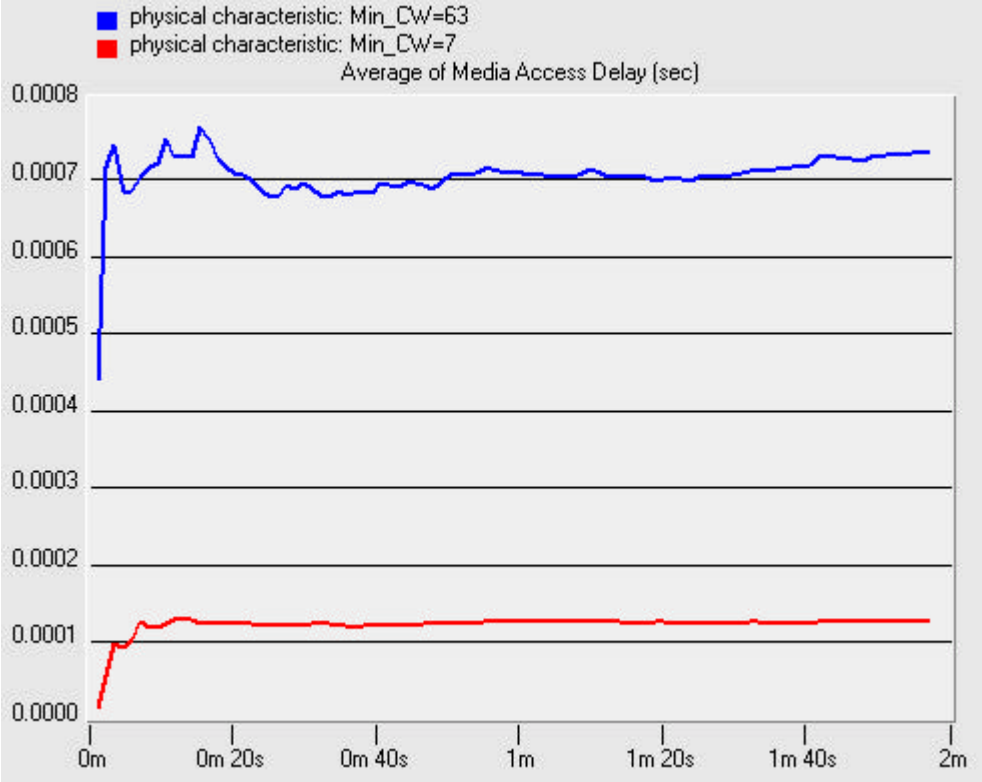

Figure 3.4 Simulation results of different Min Contention Window

From the Figure 3.4, it is easy to see that the performance of the wireless network can be improved by setting "Min Contention Window" to a smaller value in case that there are few wlan stations in the network.

PERFORMANCE TUNE-UP: WLAN PARAMETERS

There are two important wlan parameters that affect the performance of wireless LAN. They are "Fragmentation threshold" and "RTS/CTS threshold". "RTS/CTS threshold" is mainly used to deal the hidden terminal problem and "Fragmentation threshold" is used to improve the wlan performance when the media error rate is high. In this part of the project, I will simulate and study the effect of the Fragmentation Threshold on the WLAN performance.

4.1 Implementation

Because Fragmentation of packet is useful when the wlan media error rate is high, so to study the effect of fragmentation threshold, it is needed to introduce Bits Error Rate (BER) into the OPNET WLAN model.

Because the wireless channel in the OPNET WLAN model is error-free, I developed a "Packet Error Generator" and integrated it into the wlan_station model. (Figure 4.1)

The Packet Error Generator (PEG) can work in three modes:
1. "Disabled": in this mode, PEG will not introduce any error into the wireless channel.
2. "Bit Error Mode": in this mode, PEG will count and calculate the total bits received from all other stations. Once that number reaches the specified "bit error rate" threshold, PEG will damage the current packet and report the loss to the MAC layer.
3. "Packet Error Mode": in this mode, PEG will count the total number of packets received from all other stations. Once that number reaches the specified "packet error rate" threshold, PEG will damage the current packet and report the loss to the MAC layer.

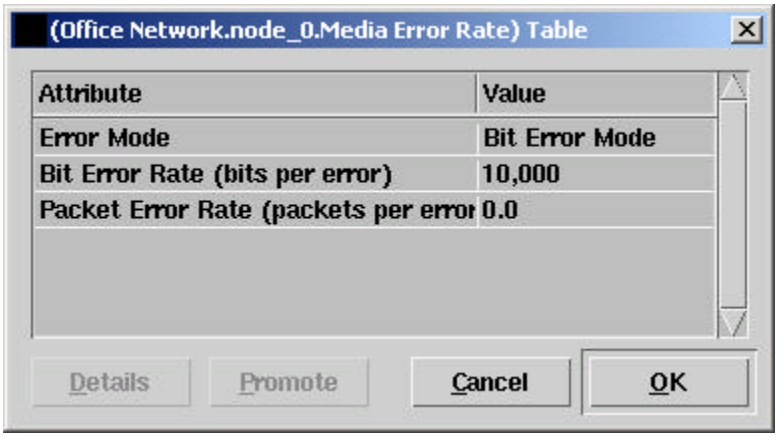

Figure 4.1: Packet Error Generator

3.2 Scenario and Settings

In the simulation, a simple scenario with two wlan stations is used (Figure 4.2). The two stations send data to each other at an average rate of about 820Kbits/s during the simulation. Traffic type is listed in Table 4.1 in detail.

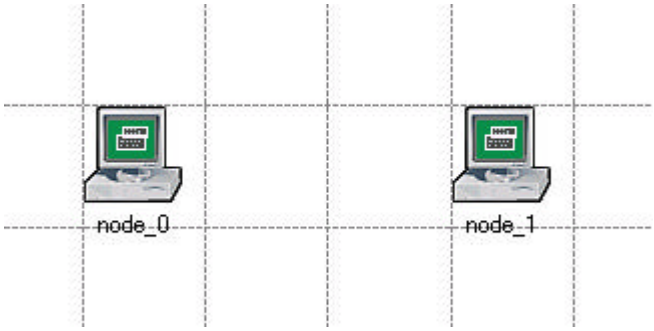

Figure 4.2: Scenario for Simulation

Attribute	Value
Time start to send packets (second)	constant (2)
Time stop sending packets (second)	Never
Packet interarrival time (second)	exponential (0.01)
Packet size (bytes)	exponential (1024)
Segmentation size (bytes)	No segmentation

Table 4.1 Traffic generation parameters

3.3 Simulation, Results and Analysis

Nine sets of simulation with different combination of Bits Error Rate and Fragmentation Threshold are done to study the effectiveness of fragmentation threshold. Parameters for the simulation set 1~9 are listed in Table 4.2 Table 4.3 and Table 4.4.

In the simulation, *Throughput* of node_0 is collected for analysis.

Throughput*: In OPNET, Throughput has the specific meaning of "The bits rate sent to the higher layer" It is a representation of the rate of data successfully received from all other stations.*

Simulation sets 1-2-3:

	Simulation set 1	Simulation set 2	Simulation set 3
Error Mode	Bit Error Mode	Bit Error Mode	Bit Error Mode
Bits Error Rate (1/bits)	1/50,000	1/50,000	1/50,000
Fragmentation Threshold	None	256 bytes	512 bytes
WLAN bandwidth (bps)	11M	11M	11M
WLAN buffer size (bits)	256K	256K	256K
Max receive lifetime (s)	0.5	0.5	0.5
Short retry limit (slots)	7	7	7
Long retry limit (slots)	4	4	4
PHY characteristic	Frequency hopping	Frequency hopping	Frequency hopping

Table 4.2 Parameters of Simulation set 1, 2 and 3 on Fragmentation Threshold

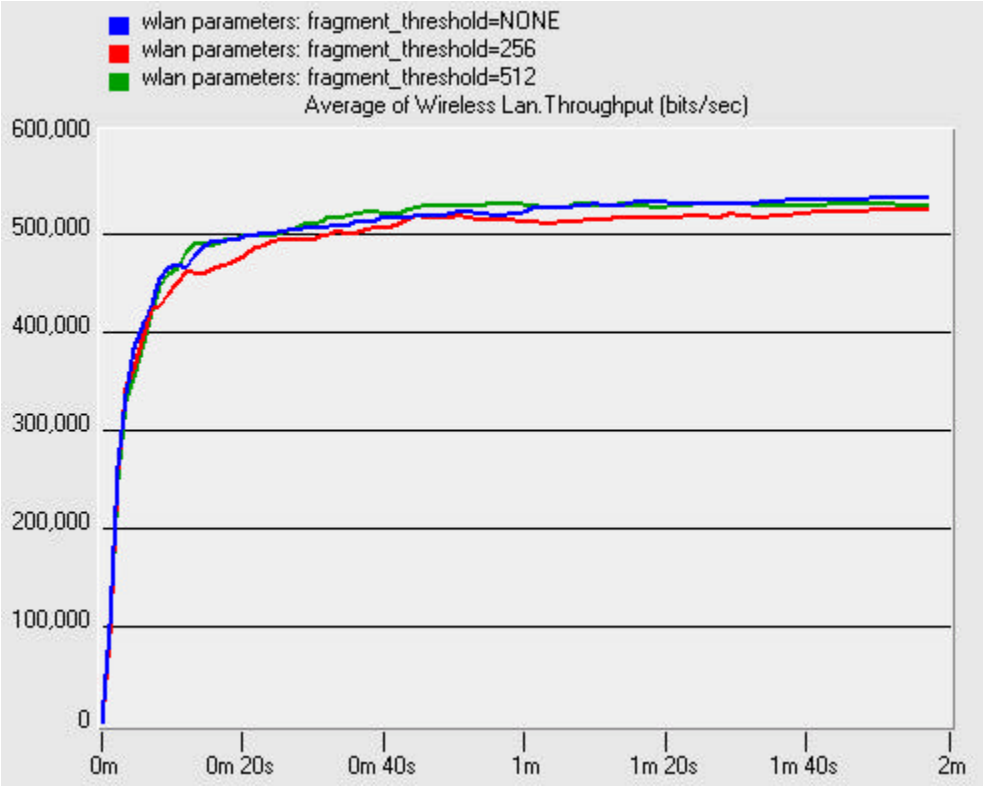

Figure 4.3: Simulation results of simulation set 1,2 and 3

From Figure 4.3, we can see that when the bits error rate is low (1/50,000), the different fragmentation thresholds (256 bytes or 512 bytes) or even without fragmentation (NONE) do not affect the performance of wlan a lot.

Simulation sets 4-5-6:

	Simulation set 4	**Simulation set 5**	**Simulation set 6**
Error Mode	Bit Error Mode	Bit Error Mode	Bit Error Mode
Bits Error Rate (1/bits)	1/10,000	1/10,000	1/10,000
Fragmentation Threshold	None	256 bytes	512 bytes
WLAN bandwidth (bps)	11M	11M	11M
WLAN buffer size (bits)	256K	256K	256K
Max receive lifetime (s)	0.5	0.5	0.5
Short retry limit (slots)	7	7	7
Long retry limit (slots)	4	4	4
PHY characteristic	Frequency hopping	Frequency hopping	Frequency hopping

Table 4.3 Parameters of Simulation set 4, 5 and 6 on Fragmentation Threshold

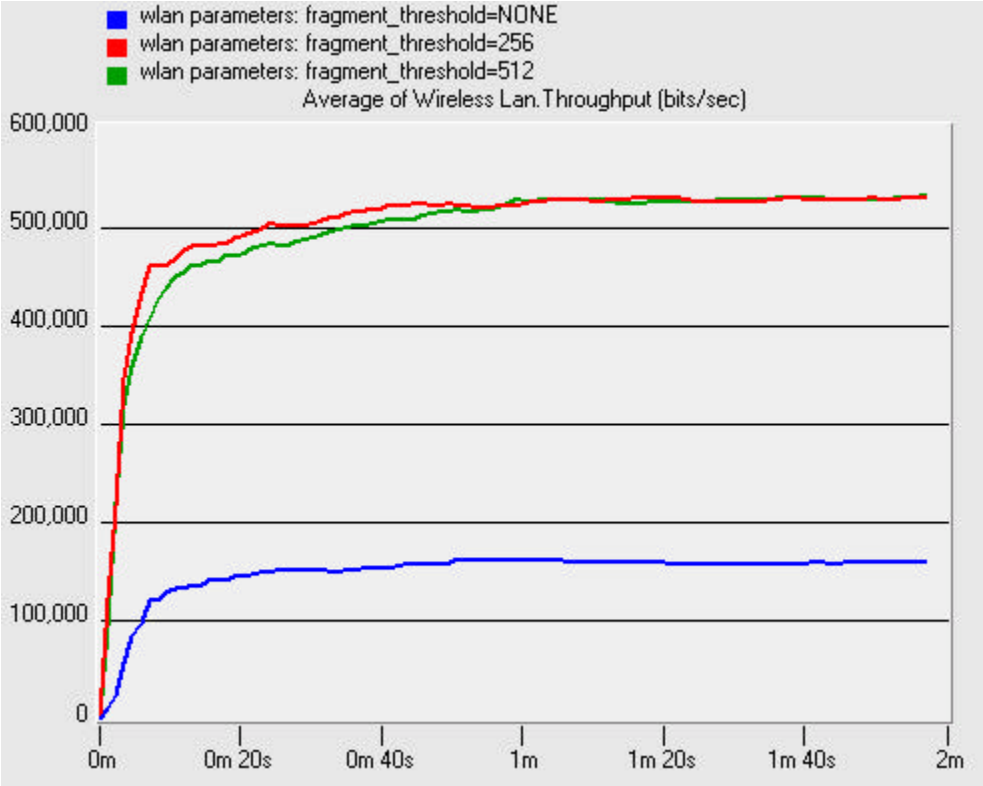

Figure 4.4: Simulation results of simulation set 4,5 and 6

From Figure 4.4, we can see that when the bits error rate is relatively high (1/10,000), the fragmentation threshold (256 bytes and 512 bytes) can achieve a much better performance than without using a fragmentation threshold.

Simulation sets 7-8-9:

	Simulation set 7	**Simulation set 8**	**Simulation set 9**
Error Mode	Bit Error Mode	Bit Error Mode	Bit Error Mode
Bits Error Rate (1/bits)	1/500,000	1/500,000	1/500,000
Fragmentation Threshold	None	16 bytes	256 bytes
WLAN bandwidth (bps)	11M	11M	11M
WLAN buffer size (bits)	256K	256K	256K
Max receive lifetime (s)	0.5	0.5	0.5
Short retry limit (slots)	7	7	7
Long retry limit (slots)	4	4	4
PHY characteristic	Frequency hopping	Frequency hopping	Frequency hopping

Table 4.4 Parameters of Simulation set 7, 8 and 9 on Fragmentation Threshold

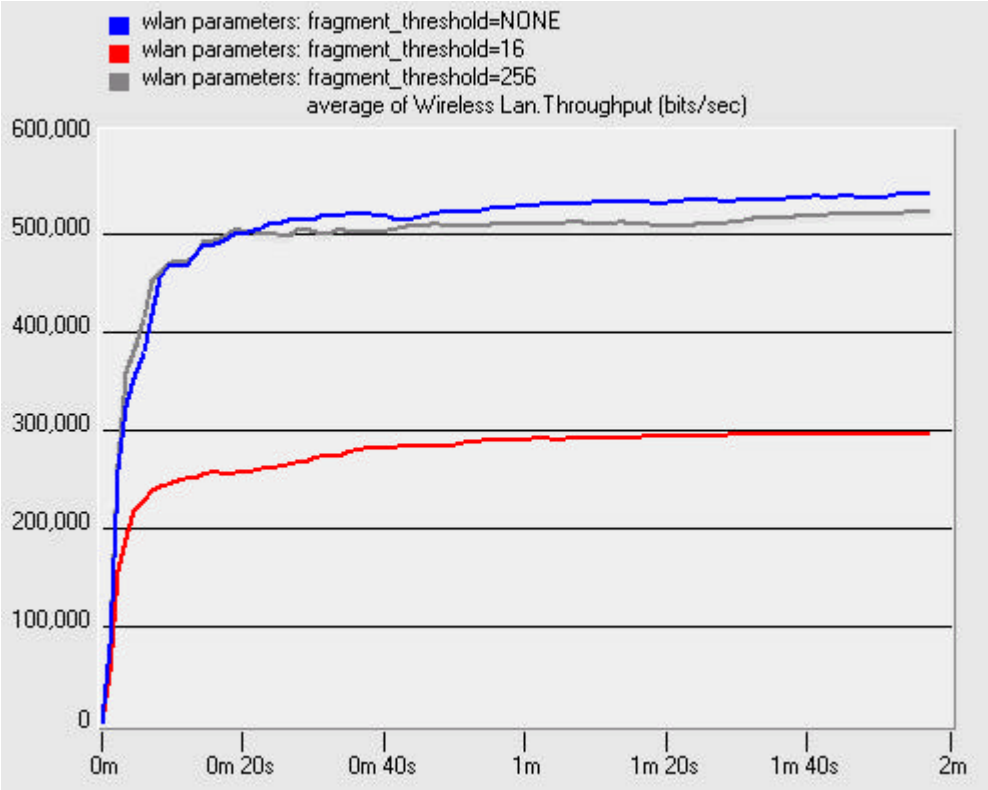

Figure 4.5: Simulation results of simulation set 7,8 and 9

From Figure 4.5, we can see that when the bits error rate is relatively low (1/500,000), there is nearly no difference between using a normal fragmentation threshold (256 bytes) and not using a fragmentation threshold. But if we set the fragmentation threshold to a very small value (16 bytes), the performance of wlan will be greatly reduced because the packet header will occupy too much bandwidth in this case.

PERFORMANCE TUNE-UP: ADAPTIVE BACK-OFF

In this part of the project, I implement and simulate a new adaptive back-off mechanism that is described in [1]. This mechanism is named Distributed Contention Control (DCC) by the author and it is for the adaptive reduction of contention in Wireless LAN that utilizes random access MAC protocols. This mechanism can be executed on the top of pre-existent access scheduling protocol (DCF) and does not introduce any additional overhead.

The main idea of this adaptive back-off mechanism is to estimate the shared channel's contention level by calculating the slots utilization rate (Figure 5.1). If a wlan station detects that the contention level of the shared channel is high, which means that it is very possible a collision will happen if the station send out the packet now, it will trigger the Virtual Collision Procedure and do the back-off immediately instead of sending out the packet. Thus, a possible collision is

avoided.

For more information about the adaptive back-off algorithm, please refer to [1][2][3][4] in the references section.

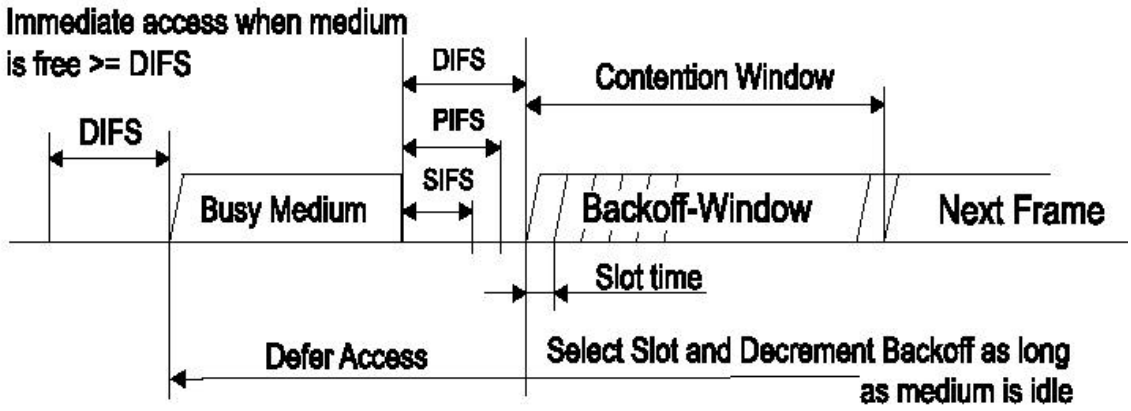

Figure 5.1 Basic media access method

5.1 Implementation

The adaptive back-off mechanism is implemented and integrated into the wlan_mac process model (Figure 5.2). Two states (PT_TEST and PT_BACKOFF) and one condition (PT_SATISFIED) are inserted into the process model. States IDLE, DEFFER, BACKOFF_NEEDED, BACKOFF and TRANSMIT are modified. Interruptions are modified in the Function Block. Also, two options are added to the Node Attributes Interface for easy switching between "Standard Backoff" and "Adaptive Backoff" mode (Figure 5.3).

For more information of the implementation, please refer to the Appendix Section.

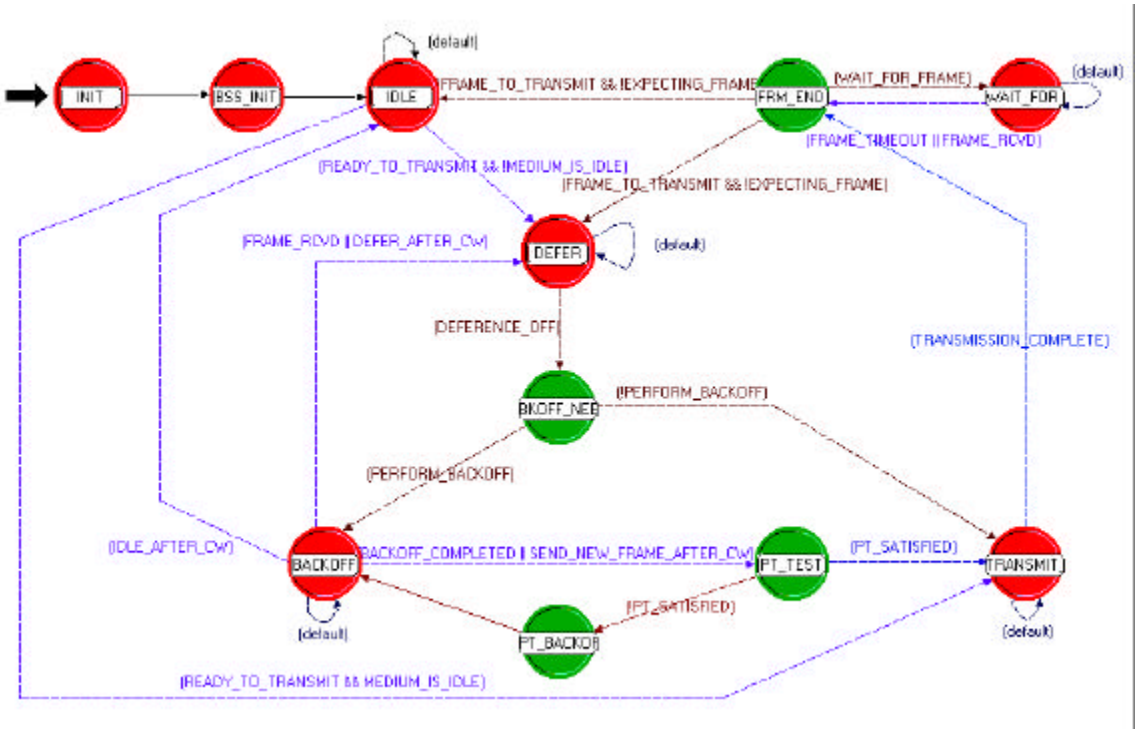

Figure 5.2 Modified wlan_mac process model

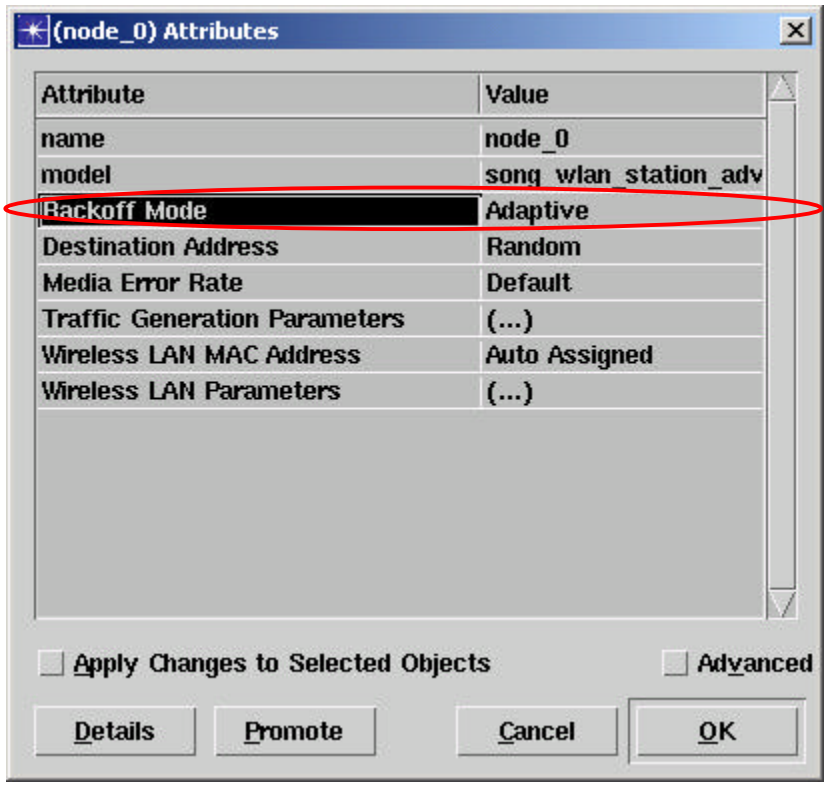

Figure 5.3: Node Attributes Interface

5.2 Scenario and Settings

Three scenarios are used to do the simulation. The scenarios are similar except for the different number of wlan stations in them (11 wlan stations in the first scenario, 21 wlan stations in the second scenario and 65 wlan stations in the third scenario). All stations have the same behavior. They are sending data to other stations with an average rate of about 820Kbits/s during the simulation. Destination stations are chosen randomly by the source station. Traffic type is listed in Table 5.1 in detail.

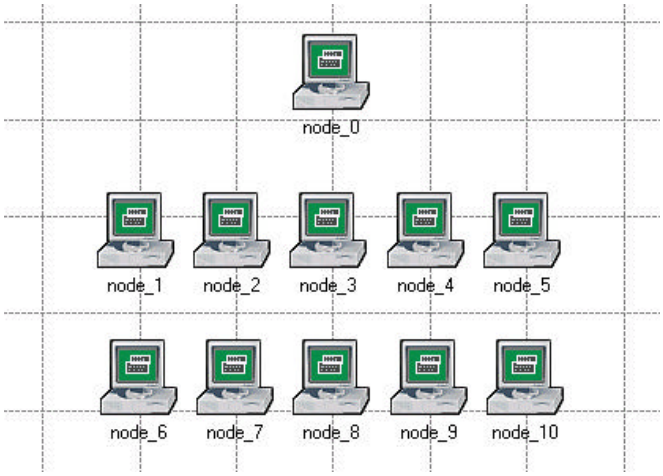

Figure 5.4: Simulation scenario with 11 stations

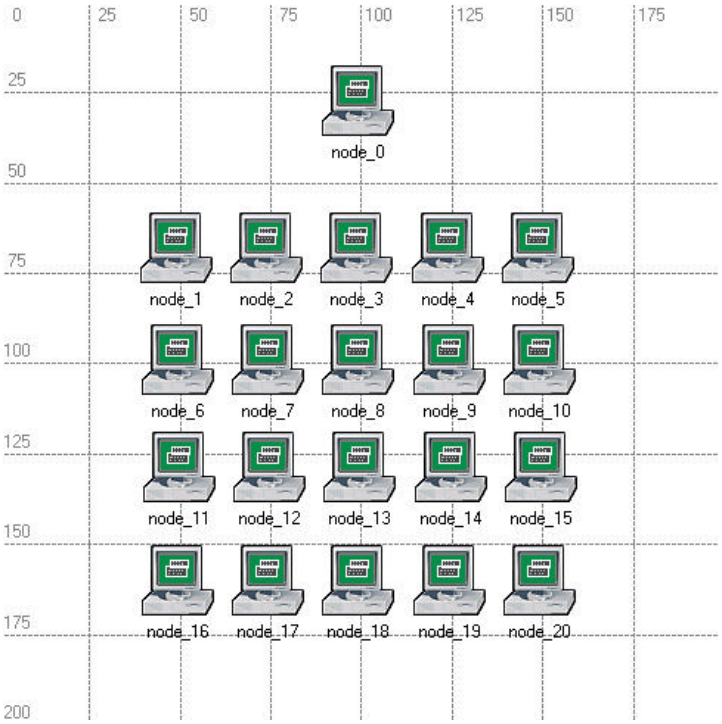

Figure 5.5: Simulation scenario with 21 stations

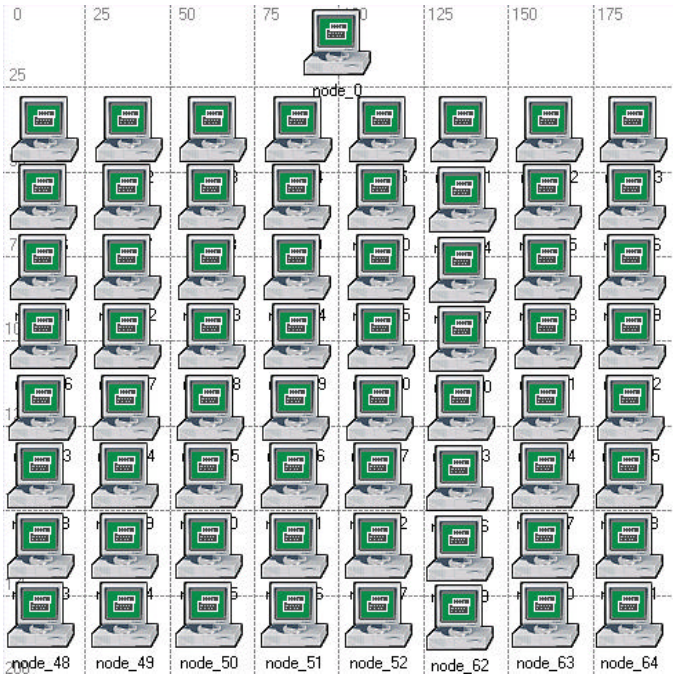

Figure 5.6: Simulation scenario with 65 stations

Attribute	Value
Time start to send packets (second)	constant (2)
Time stop sending packets (second)	Never
Packet interarrival time (second)	exponential (0.01)
Packet size (bytes)	exponential (1024)
Segmentation size (bytes)	No segmentation

Table 5.1 Traffic generation parameters

5.3 Simulation, Results and Analysis

The simulation parameters corresponding to Scenario 1~3 are listed in Table 5.2 as simulation sets 1~3.

In the simulation, *Throughput* and *Load* of node_0 is collected for analysis.

Throughput*: In OPNET, Throughput has the specific meaning of "The bits rate sent to the higher layer" It is a representation of the rate of data successfully received from all other stations.*
Load of wlan: *In OPNET, "Load" has the specific meaning of "The bits rate submitted to wlan layer by all other higher layers in this node." It's also a representation of the rate of data send*

out to all other stations.

	Simulation set 1	**Simulation set 2**	**Simulation set 3**
Number of stations	11	21	65
Error Mode	None	None	None
Bits Error Rate (1/bits)	N/A	N/A	N/A
Fragmentation Threshold	None	None	None
WLAN bandwidth (bps)	11M	11M	11M
WLAN buffer size (bits)	256K	256K	256K
Max receive lifetime (s)	0.5	0.5	0.5
Short retry limit (slots)	7	7	7
Long retry limit (slots)	4	4	4
PHY characteristic	Frequency hopping	Frequency hopping	Frequency hopping

Table 5.2 Parameters of Simulation set 1, 2 and 3

Simulation Results:

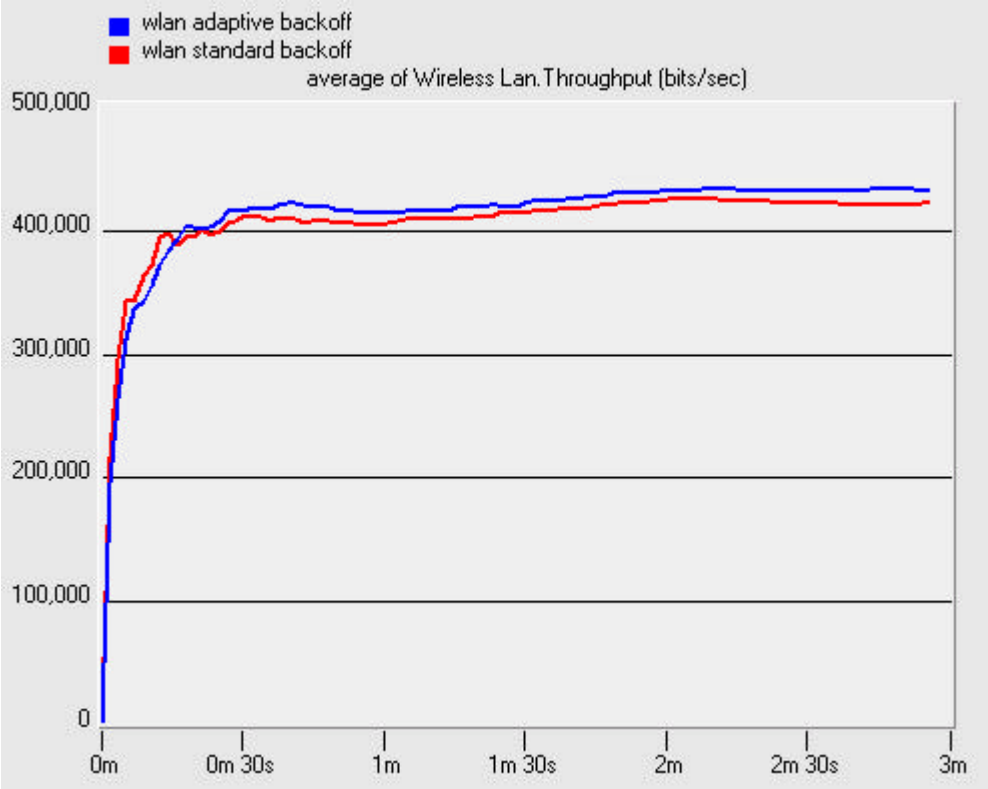

Figure 5.7: Simulation results of wlan Throughput with 11 stations

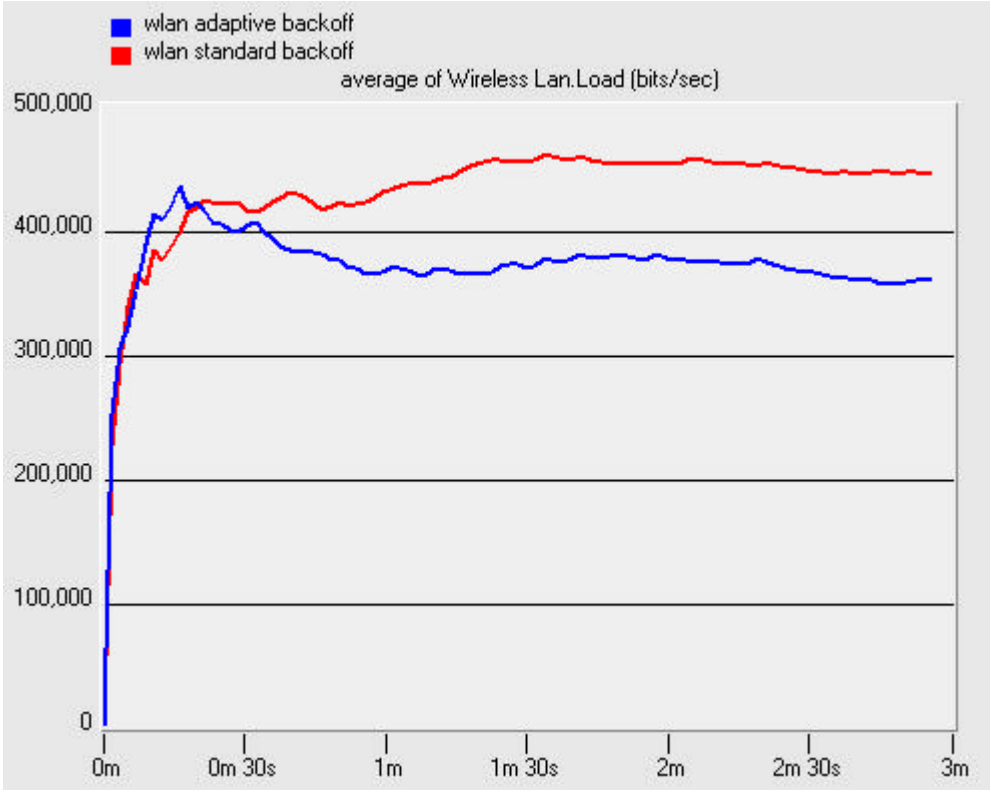

Figure 5.8: Simulation results of wlan Load with 11 stations

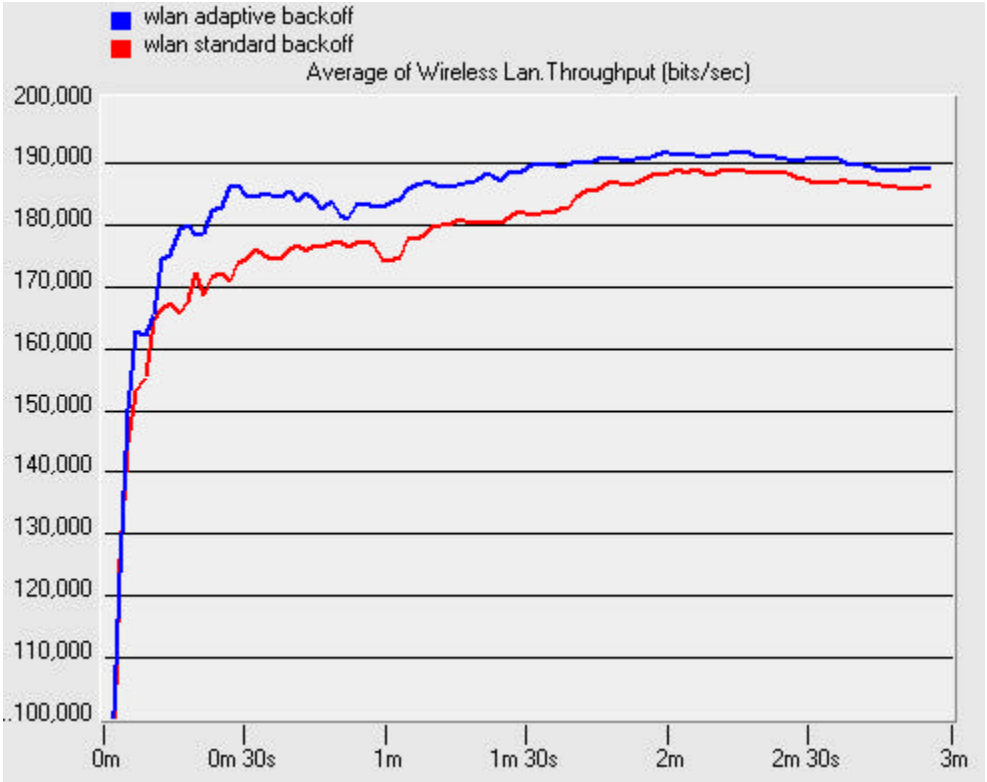

Figure 5.9: Simulation results of wlan Throughput with 21 stations

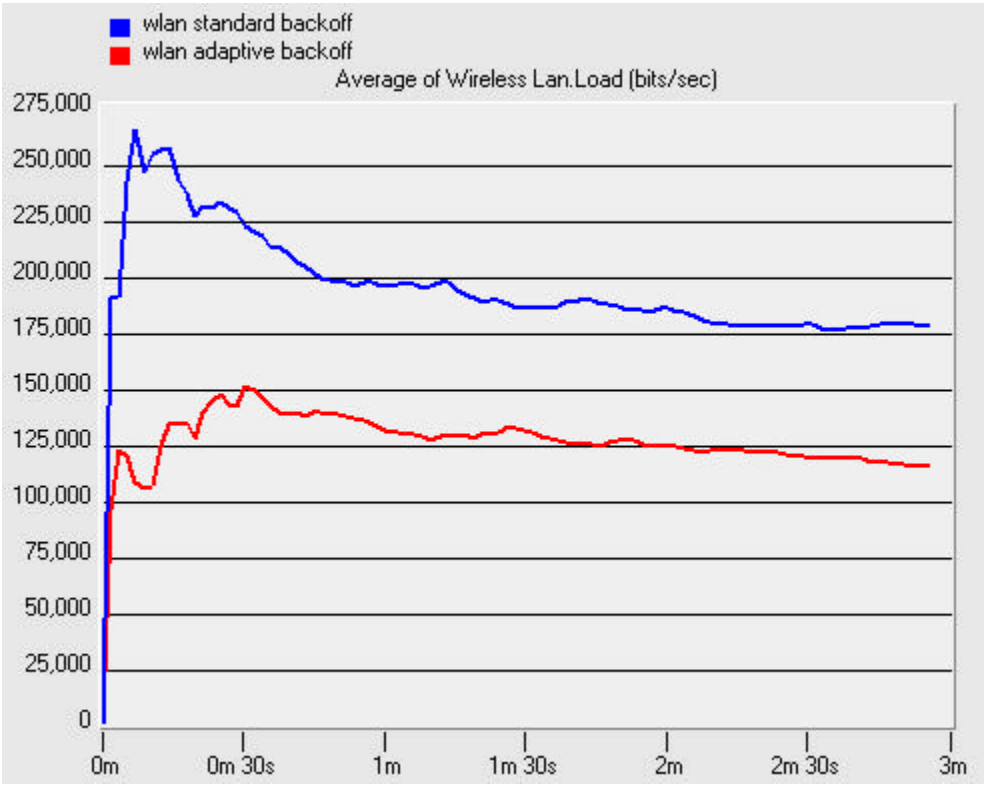

Figure 5.10: Simulation results of wlan Load with 21 stations

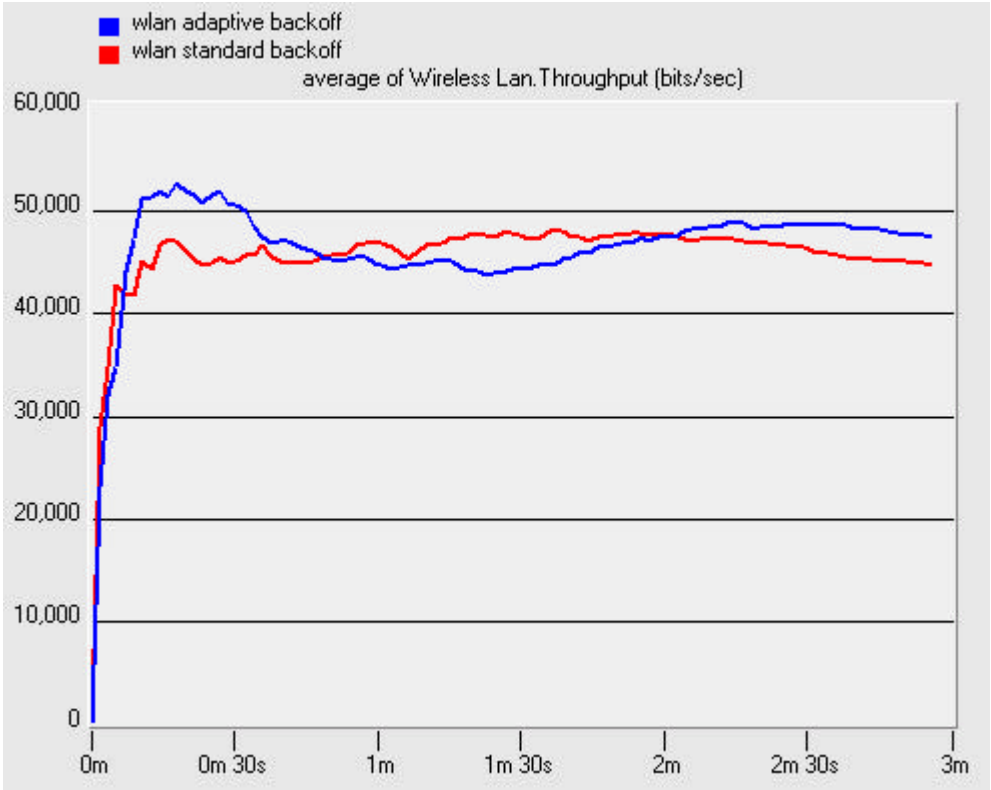

Figure 5.11: Simulation results of wlan Throughput with 65 stations

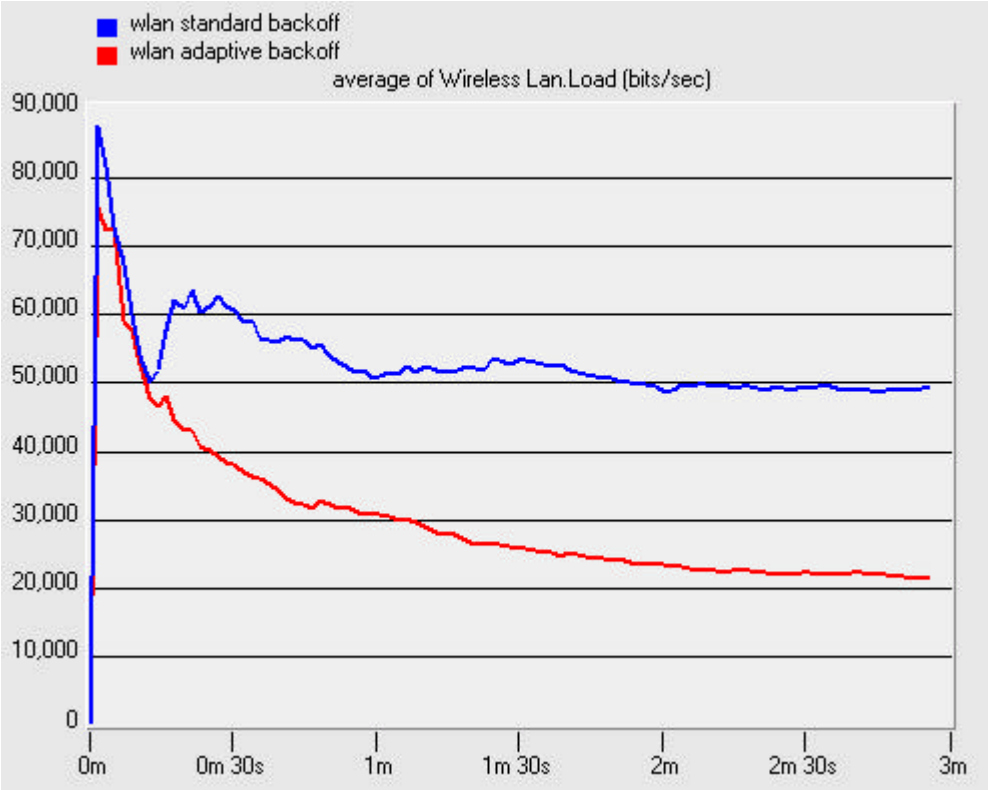

Figure 5.12: Simulation results of wlan Load with 65 stations

From Figure 5.7 ~ Figure 5.12, it is easy to see that with adaptive back-off mechanism, wlan station can greatly reduce the *wlan Load* while still maintain the same or achieve a slightly higher *wlan throughput*. The reduction of *wlan Load* is especially useful for the power saving issue of wireless devices. These results also indicate that the adaptive back-off mechanism can effectively reduce the collision and data loss in the wireless networks.

PERFORMANCE TUNE-UP: SMART SNOOP*

Smart snoop is a TCP-aware link-layer scheme for performance enhancement of wlan. It makes the lossy link appear as a higher quality link with a reduced effective bandwidth. The smart snoop is based on the Snoop protocol and uses a SMART (Simple Method to Aid Retransmissions) strategy, which combines the best feature of Go-Back-N and Selective Acknowledgement. For more information about the smart snoop protocol, please refer to [5][6][17] in the references section.

In last year's wonderful project "Performance of TCP Protocol Running over WLAN 802.11 with the Snoop Protocol"[17], Jack Chi-Kit Chow and Chi-ho Ng implemented the snoop protocol in OPNET wlan_workstation node model. In this part of my project, I am trying to implement the smart snoop in the OPNET wireless_router node model (Figure 6.1).

Instead of inserting new process mode into the node model, I am trying to integrate the Smart snoop into the ARP (Address Resolution Protocol). The Smart Snoop will be enabled in the Wireless LAN side (Figure 6.2, in the red circle).

Because of the considerably large amount of work needed in this part and I work on the whole project by myself, this part is taken as an additional and tentative part for study purpose. It is yet not finished by far.

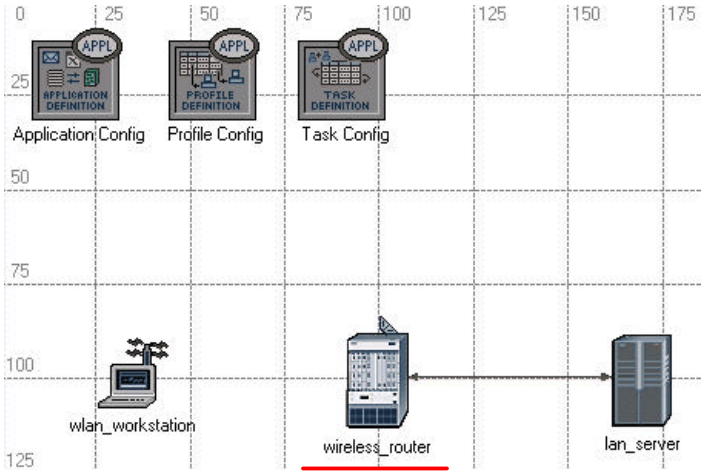

Figure 6.1: Scenario for SMART Snoop

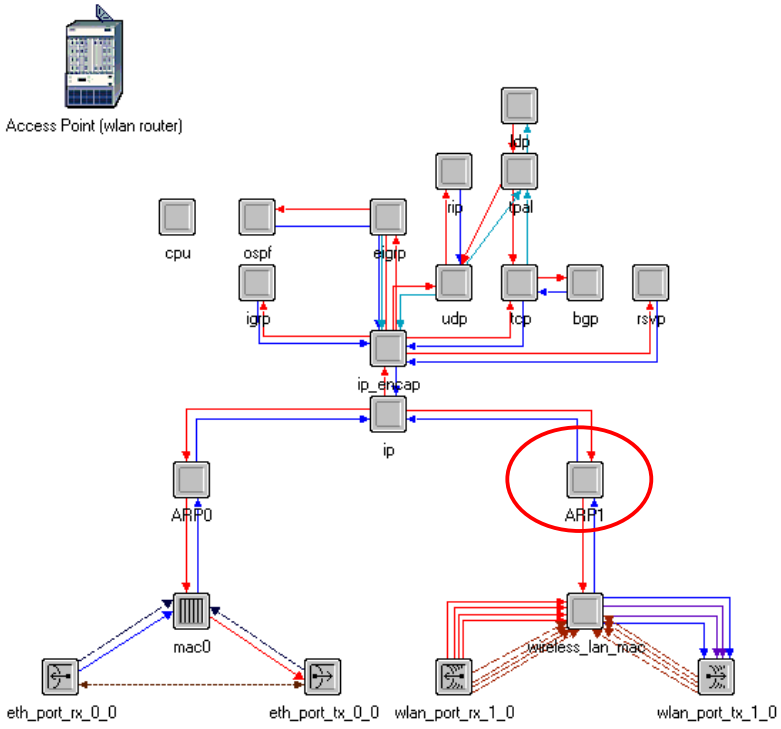

Figure 6.2: Smart snoop protocol will be integrated in ARP1

CONCLUSION AND DISCUSSION

In this project, OPNET wlan_mac process model is modified to add the customized PHY parameters; a packet error generator is implemented and added into the wlan_station node model; a new adaptive back-off algorithm is implemented and integrated into the wlan_mac process model; smart snoop protocol is studied and partly implemented; a survey of methods for improving wlan performance is done; simulations are done on three approaches for improving wlan performance; simulation results are gathered and interpreted; several suggested future work in last year's project [18] are accomplished.

Conclusions:

- ✧ Tuning-up the PHY characteristic related parameters, such as Slot time and SIFS, can greatly improve the WLAN performance, but it requires the support from hardware.
- ✧ Tuning-up the PHY characteristic related parameters, such as Minimum Contention Window, can also improve the WLAN performance.
- ✧ A properly set WLAN parameter, such as Fragmentation Threshold, can greatly increase the WLAN performance when the channel bits error rate is high.
- ✧ An adaptive back-off algorithm on the MAC layer can effectively reduce the collision in the wireless network and can also save power for wireless devices without harming the WLAN performance.

Difficulties:

- ✧ To implement and integrate my own components (such as Packet Error Generator, Adaptive Back-off mechanism) into existing OPNET models, I have to read and understand both the IEEE802.11 standard and the OPNET WLAN process model source code.
- ✧ Because OPNET does not use an Object Oriented Programming approach, debugging with OPNET is difficult. For example, it took me two whole days to find out one bug (commented out in Appendix, Page xiii).
- ✧ Without a team member, I could not discuss some very specific problems during the programming period and I have ever gone to the dead end once or two.

Suggested Future Works:

- ✧ Implement and integrate Smart Snoop protocol into the OPNET wireless_router node model
- ✧ Implement the adaptive back-off algorithm described in [2][3][4]
- ✧ Research the Hidden Terminal Problem
- ✧ More comprehensive simulations

ACKNOWLEDGEMENT

Thanks Prof. Ljiljana Trajkovic for the advices, which are greatly helpful to this project.
Thanks TA Bruce Chen and Wen Jin for all the help throughout the whole semester.

REFERENCES

[1] Luciano Bononi , Marco Conti , Lorenzo Donatiello "Design and performance evaluation of a distributed contention control(DCC) mechanism for IEEE 802.11 wireless local area networks" Proceedings of first ACM international workshop on Wireless mobile multimedia October 1998

[2] Cali, F.; Conti, M.; Gregori, E. "IEEE 802.11 protocol: design and performance evaluation of an adaptive backoff mechanism" Selected Areas in Communications, IEEE Journal on , Volume: 18 Issue: 9 , Sept. 2000

[3] Cali, F.; Conti, M.; Gregori, E. "IEEE 802.11 wireless LAN: capacity analysis and protocol enhancement" INFOCOM '98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, Volume: 1, 1998

[4] Cali, F.; Conti, M.; Gregori, E. "Dynamic tuning of the IEEE 802.11 protocol to achieve a theoretical throughput limit", IEEE/ACM Transactions on Networking, Volume: 8 Issue: 6 Dec. 2000

[5] Hari Balakrishnan , Venkata N. Padmanabhan , Srinivasan Seshan , Randy H. Katz , "A comparison of mechanisms for improving TCP performance over wireless links" , IEEE/ACM Transactions on Networking (TON) December 1997 Volume 5 Issue 6

[6] S. Keshav and S. Morgan. "SMART Retransmission:Performance with Overload and Random Losses" In Proc. Infocom' 97, 1997

[7] N. Alborz , Lj. Trajkovic, "Implementation of VirtualClock scheduling algorithm in OPNET," OPNETWORK 2001, Washington, DC, Aug. 2001

[8] G. Anastasi , L. Lenzini "QoS provided by the IEEE 802.11 wireless LAN to advanced data applications" Wireless Networks March 2000 Volume 6 Issue 2

[9] Song Ci , Hamid Sharif , Guevara Noubir "Improving performance of MAC layer by using congestion control/avoidance methods in wireless network" Proceedings of the 16th ACM SAC2001 symposium on Applied computing March 2001

[10] Arun K. Somani , Indu Peddibhotla, "Experimental evaluation of throughput performance of IRTCP under noisy channels", Proceedings of the second ACM international workshop on Wireless mobile multimedia August 1999

[11] George Xylomenos, George C. Polyzos, Petri Mahnen, Mika Saaranen "TCP Performance Issues over Wireless Links" IEEE Network, July-August 1999

[12] G. Xylomenos and G. C. Polyzos, "TCP and UDP Performance over a Wireless LAN" Proceedings of the IEEE INFOCOM 99 Conference, 1999

[13] C. Parsa and J.J. Garcia-Luna-Aceves, "TULIP: A Link-Level Protocol for Improving TCP over Wireless Links" Proc. IEEE Wireless Communications and Networking Conference 1999

[14] Andrew Muir , J. J. Garcia-Luna-Aceves "An efficient packet sensing MAC protocol for wireless networks" Mobile Networks and Applications August 1998 Volume 3 Issue 2

[15] Rodrigo Garces , J. J. Garcia-Luna-Aceves "Collision avoidance and resolution multiple access with transmission queues" Wireless Networks March 1999 Volume 5 Issue 2

[16] B. Rathke, M. Schlager, and A. Wolisz, "Systematic measurement of TCP performance over wireless LANs" Tech. Rep., Technical University of Berlin, Germany, 1998

[17] Jack Chi-Kit Chow, Chi-ho Ng "ENSC 833 Project Report: Performance of TCP Protocol Running over WLAN 802.11 with the Snoop Protocol" Spring 2001

[18] Jim Chuang, Tim Yao-Ting Lee, Marion Sum "ENSC 833-3 Network Protocols and Performance: Wireless Ethernet performance" Project Report, spring 2001

[19] LAN MAN Standards Committee of the IEEE Computer Society "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications" ANSI/IEEE Std 802.11, 1999 Edition

[20] OPNET technologies, Inc., "Wireless LAN Model Description" and various OPNET online help file and documents. http://www.opnet.com/products/library/WLAN_Model_Guide1.pdf

APPENDIX: CODE LISTING

```
//*********************************************************************************************//
// James Song songs@acm.org     May 2002
// Note:
// Code segments added by myself are commented in this format.
// Complete project file can be downloaded from
// http://www.songssoft.com/james/courses/cmpt885/project/project.zip
//*********************************************************************************************//
```

song_wlan_mac.header block

```
//*********************************************************************************************//
// James Song  songs@acm.org      May 2002
typedef enum WlanT_Error_Mode_Char_Code
      {
      Error_Disabled,
      Bit_Error_Mode,
      Packet_Error_Mode
      } WlanT_Error_Mode_Char_Code;
//*********************************************************************************************//

//*********************************************************************************************//
// James Song  songs@acm.org      May 2002
typedef struct WlanT_Err_Data_Struct
      {
      double packets_received;
      double bits_received;
      double packets_damaged;
      double cur_packets;
      double cur_bits;
      } WlanT_Err_Data_Struct;
//*********************************************************************************************//

//*********************************************************************************************//
// James Song  songs@acm.org      May 2002
typedef struct WlanT_PT_Struct
      {
      Boolean trans_flag;
      Boolean vb_flag;
      Boolean debug_flag;
      double slot_utilization;
```

```
        double busy_slots;
        double init_backoff_slots;
        double p_trans;
        int attempts;
        } WlanT_PT_Struct;
//*****************************************************************************************//

///*****************************************************************************************//
// James Song  songs@acm.org       May 2002
#define PT_SATISFIED                    (p_t.trans_flag == OPC_TRUE)
///*****************************************************************************************//

///*****************************************************************************************//
// James Song  songs@acm.org       May 2002
static void                 pt_backoff();
///*****************************************************************************************//
```

song_wlan_mac.state variable

```
/* all packets : bad packets #by James Song */
double      \packet_error_rate;

/* all bits : error bits #by James Song */
double      \bit_error_rate;

/* type the error mode introduced into the process model #James Song */
WlanT_Error_Mode_Char_Code      \media_error_mode;

/* Data sturcture to store the error control data #James Song */
WlanT_Err_Data_Struct   \err_data;

/* Backoff mode used in the mac layer #James Song */
int     \backoff_mode;

/* Possibility of transmission data set #James Song */
WlanT_PT_Struct     \p_t;

Stathandle      \slots_utilization_handle;

Stathandle      \possibility_transmission_handle;
```

song_wlan_mac.temporary variables

```
int                             p; //James Song
double                          q; //James Song
```

BKOFF_NEEDED: Enter Execs

```
/* Checking wether backoff is needed or not     */
if     (wlan_flags->backoff_flag     ==     OPC_BOOLINT_ENABLED     ||     wlan_flags->perform_cw     ==
OPC_BOOLINT_ENABLED)
      {
      if (backoff_slots == 0)
            {
            /* 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_BOOLINT_ENABLED)
                  {
                  /* If retry count is set to 0 then set the maximum backoff  */
                  /* slots to min window size.                                */
                  max_backoff = cw_min;
                  }
            else
                  {
                  /* We are restransmitting. 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));

            //*****************************************************************************************//
            // James Song  songs@acm.org       May 2002
            if (backoff_mode == 1)    //adaptive backoff
                  {
```

```
                p_t.trans_flag = OPC_TRUE;
                p_t.slot_utilization = 0;
                p_t.init_backoff_slots = backoff_slots;
                p_t.busy_slots = 0;
                p_t.p_trans = 1;
                p_t.attempts = retry_count;
                }
        //*****************************************************************************************//
        }

    /* Set a timer for the end of the backoff interval.                    */
    intrpt_time = (current_time + backoff_slots * slot_time);

    /* Scheduling self interrupt for backoff.                                  */
    if (wlan_flags->perform_cw == OPC_BOOLINT_ENABLED)
        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);
    }
```

BACKOFF: Exit Execs

```
/* Call the interrupt processing routine for each interrupt.        */
wlan_interrupts_process ();

/* Set the number of slots to zero, once the backoff is completed.    */
if (BACKOFF_COMPLETED)
    {
    backoff_slots = 0.0;
    }
else if (CW_COMPLETED)
    {
    backoff_slots = 0.0;

    /* Reset the contention window flags to enable future                  */
    /* transmissions.                                                      */
    wlan_flags->cw_required = OPC_BOOLINT_DISABLED;
    wlan_flags->perform_cw    = OPC_BOOLINT_DISABLED;
    }
```

```
/* Storing remaining backoff slots if the frame is rcvd from the */
/* remote station.                                                */
if (RECEIVER_BUSY_HIGH)
	{
	/* Computing remaining backoff slots for next iteration.  */
	backoff_slots =  ceil ((intrpt_time - current_time) / slot_time);

	//*************************************************************************************//
	// James Song  songs@acm.org	May 2002
	if (backoff_mode == 1)  //adaptive backoff
	{
	p_t.busy_slots = p_t.busy_slots + 1;
	}
	//*************************************************************************************//

	if (op_ev_valid (backoff_elapsed_evh) == OPC_TRUE)
		{
		/* Clear the self interrupt as station needs to defer.  */
		op_ev_cancel (backoff_elapsed_evh);

		/* Disable perform cw flag because the station will no  */
		/* backoff using contention window.                     */
		wlan_flags->perform_cw  = OPC_BOOLINT_DISABLED;
		}
	}

/* Schedule deference if the frame is received while the station is  */
/* backing off.                                                      */
if (FRAME_RCVD || DEFER_AFTER_CW)
	{
	wlan_schedule_deference ();
	}
```

PT_TEST: Enter Execs

```
//*************************************************************************************//
// James Song  songs@acm.org	May 2002
if (backoff_mode == 1)  //adaptive backoff
	{

		if (p_t.init_backoff_slots > 0.5) //p_t.init_backoff_slots > 0
```

```
                {
                p_t.slot_utilization = p_t.busy_slots / p_t.init_backoff_slots;

                p = 0;
                q = p_t.slot_utilization;

                p_t.attempts = retry_count;

                while (p < p_t.attempts)
                {
                q = q * p_t.slot_utilization;
                p++;
                }

                p_t.p_trans = 1 - q;

                if (p_t.p_trans > op_dist_uniform (1))
                      {
                      p_t.trans_flag = OPC_TRUE;
                      p_t.vb_flag = OPC_FALSE;
                      }
                else   //virtual collision and backoff again
                      {
                      p_t.trans_flag = OPC_FALSE;
                      p_t.vb_flag = OPC_TRUE;
                      }

                op_stat_write (possibility_transmission_handle, p_t.p_trans);
                op_stat_write (slots_utilization_handle, p_t.slot_utilization);

                }
          else   //p_t.init_backoff_slots == 0
                {
                p_t.trans_flag = OPC_TRUE;
                p_t.vb_flag = OPC_FALSE;
                op_stat_write (possibility_transmission_handle, 1);
                op_stat_write (slots_utilization_handle, 0);
                }
     }

else   //ieee 802.11 standard backoff
```

```
        {
        p_t.trans_flag = OPC_TRUE;
        p_t.vb_flag = OPC_FALSE;
        }

//*****************************************************************************************//
```

PT_BACKOFF: Enter Execs

```
//*****************************************************************************************//
// James Song  songs@acm.org       May 2002
//static void pt_backoff()
//      {
        char debug_msg [100];

//      FIN (pt_backoff());

        wlan_flags->backoff_flag = OPC_BOOLINT_ENABLED;
        wlan_flags->perform_cw == OPC_BOOLINT_DISABLED;
        //wlan_flags->wait_eifs_dur == OPC_BOOLINT_ENABLED;

        retry_count = retry_count + 1;

        // Compute backoff interval using binary exponential process.
        // After a successful transmission we always use cw_min.
        if (retry_count == 0)
                {
                // If retry count is set to 0 then set the maximum backoff
                // slots to min window size.
                max_backoff = cw_min;
                }
        else
                {
                // We are restransmitting. 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));

        p_t.trans_flag = OPC_TRUE;
        p_t.slot_utilization = 0;
        p_t.init_backoff_slots = backoff_slots;
        p_t.busy_slots = 0;
        p_t.p_trans = 1;
        p_t.attempts = retry_count;

        // Set a timer for the end of the backoff interval.
        intrpt_time = (current_time + backoff_slots * slot_time);

        // Scheduling self interrupt for backoff.
        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);

        //op_sim_message("pt_backoff", "####################");

//      FOUT;
//}
//*********************************************************************************************//
```

TRANSMIT_0: Enter Execs

```
//*********************************************************************************************//
// James Song  songs@acm.org       May 2002
if (backoff_mode == 1)    //adaptive backoff
        {
        p_t.trans_flag = OPC_TRUE;
        p_t.vb_flag = OPC_FALSE;
        p_t.slot_utilization = 0;
        p_t.busy_slots = 0;
        p_t.init_backoff_slots = 0;
        p_t.p_trans = 1;
```

```
        p_t.attempts = 0;
        }
//*******************************************************************************************//

if (wlan_flags->immediate_xmt == OPC_TRUE)
        {
        wlan_frame_transmit ();
        wlan_flags->immediate_xmt = OPC_FALSE;
        }

else if (wlan_flags->rcvd_bad_packet == OPC_BOOLINT_DISABLED &&
       intrpt_type == OPC_INTRPT_SELF)
        {
        wlan_frame_transmit ();
        }

if (wlan_trace_active)
        {
        /* Determine the current state name */
        strcpy (current_state_name, "transmit");
        }
```

IDLE: Enter Execs

```
if (wlan_trace_active)
        {
        /* Determine the current state name.*/
        strcpy (current_state_name, "idle");
        }

//*******************************************************************************************//
// James Song  songs@acm.org       May 2002
// Objective:      Initialize the p_t data unit
if (backoff_mode == 1)    //adaptive backoff
        {
        p_t.trans_flag = OPC_TRUE;
        p_t.vb_flag = OPC_FALSE;
        p_t.slot_utilization = 0;
        p_t.busy_slots = 0;
        p_t.init_backoff_slots = 0;
        p_t.p_trans = 1;
        p_t.attempts = 0;
```

```
	}
//*****************************************************************************//
```

song_wlan_mac.function block

```
static void wlan_physical_layer_data_arrival ()
	{
	char                                    msg_string [120];
	int                                         dest_addr;
	int                                         accept;
	int                                         data_pkt_id;
	int                                         final_dest_addr;
	WlanT_Data_Header_Fields*               pk_dhstruct_ptr;
	WlanT_Control_Header_Fields*            pk_chstruct_ptr;
	WlanT_Mac_Frame_Type                    rcvd_frame_type;
	Packet*                                     wlan_rcvd_frame_ptr;
	Packet*                                     seg_pkptr;
	double                                      cur_frame_size; //added by James Song

	/** Process the frame received from the lower layer.               **/
	/** This routine decapsulate the frame and set appropriate     **/
	/** flags if the station needs to generate a response to the **/
	/** received frame.                                                          **/
	FIN (wlan_physical_layer_data_arrival ());

	/*   Access received packet from the physical layer stream.     */
	wlan_rcvd_frame_ptr = op_pk_get (i_strm);

	op_pk_nfd_access (wlan_rcvd_frame_ptr, "Accept", &accept);

	//*****************************************************************************//
	// James Song  songs@acm.org       May 2002
	// Objective:      Test whether "Accept" field is alway set TRUE by the OPNET 'wlan_mac'
	// Result:         "Accept" is always TRUE
	// if (accept == OPC_FALSE)
	//     {
	//     op_sim_end ("Field Accept is set to FALSE");
	//     }
	//*****************************************************************************//

	//*****************************************************************************//
	// James Song  songs@acm.org       May 2002
```

```
// Objective:      Error Generator

switch (media_error_mode)
      {
      case Error_Disabled:
            {
            break;
            }
      case Bit_Error_Mode:
            {
            cur_frame_size = (double) op_pk_total_size_get (wlan_rcvd_frame_ptr);

            err_data.bits_received = err_data.bits_received + cur_frame_size;
            err_data.packets_received = err_data.packets_received + 1;

            err_data.cur_bits = err_data.cur_bits + cur_frame_size;
            err_data.cur_packets++;

            if (err_data.cur_bits > bit_error_rate)
                  {
                        err_data.cur_bits = err_data.cur_bits - bit_error_rate;
                        err_data.packets_damaged++;
                        accept = OPC_FALSE;
                  }

            break;
            }
      case Packet_Error_Mode:
            {
            cur_frame_size = (double) op_pk_total_size_get (wlan_rcvd_frame_ptr);

            err_data.bits_received = err_data.bits_received + cur_frame_size;
            err_data.packets_received = err_data.packets_received + 1;

            err_data.cur_bits = err_data.cur_bits + cur_frame_size;
            err_data.cur_packets++;

            if (err_data.cur_packets > packet_error_rate)
                  {
                        err_data.cur_packets = err_data.cur_packets - packet_error_rate;
                        err_data.packets_damaged++;
```

```
                        accept = OPC_FALSE;
                    }

                break;
                }
            default:
                {
                wlan_mac_error ("Unexpected Error Mode Characteristic encountered.",
                                OPC_NIL, OPC_NIL);
                break;
                }
            }
        //*****************************************************************************//
        //the rest part of wlan_physical_layer_data_arrival () is unchanged.
        // … …
}

static void wlan_frame_transmit ()
        {
        char                        msg_string    [120];
        char                        msg_string1 [120];
        WlanT_Hld_List_Elem*        hld_ptr;
        double                          pkt_tx_time;
        char                        debug_msg [120]; //James Song

        /** Main procedure to call functions for preparing frames.     **/
        /** The procedure to prepare frame is called in this routine **/
        FIN (wlan_frame_transmit());

        /* If Ack and Cts needs to be sent then prepare the appropriate*/
        /* frame type for transmission                                  */
        if ((fresp_to_send == WlanC_Cts) || (fresp_to_send == WlanC_Ack))
            {
            wlan_prepare_frame_to_send (fresp_to_send);

            /* Break the routine if Cts or Ack is already prepared to tranmsit */
            FOUT;
            }

        /* If it is a retransmission then check which type of frame needs to be     */
        /* retransmitted and then prepare and transmit that frame                   */
```

```
        ////////////////////////////////////////////////////////////
        //                                                        //
        //   comment out by James Song!!!!!!!!!!!!                //
        //                                                        //
        //   "else if (retry_count != 0)"                         //
        //                                                        //
        //   spent 30 hours on it :((((((((((((((                //
        //                                                        //
        ////////////////////////////////////////////////////////////

        else if ((retry_count != 0) && (wlan_transmit_frame_copy_ptr != OPC_NIL))

        //the rest part of wlan_frame_transmit () is unchanged.
        // … …
}

static void wlan_mac_sv_init ()
        {
        Objid                   mac_params_comp_attr_objid;
        Objid                   params_attr_objid;
        Objid                   chann_params_comp_attr_objid;
        Objid                   subchann_params_attr_objid;
        Objid                   chann_objid;
        Objid                   sub_chann_objid;
        Objid                   err_params_comp_attr_objid; //added by James Song
        Objid                   err_params_attr_objid; //added by James Song
        int                     num_chann;
        char                subnet_name [512];
        double                  bandwidth;
        double                  frequency;
        int                     i;
        char                 bss_name[128];
        Log_Handle              bssid_changed_log_handle;
        char                ip_addr_mode [64];

        //*********************************************************************************//
        // James Song  songs@acm.org       May 2002
        op_ima_obj_attr_get (my_objid, "Media Error Rate", &err_params_comp_attr_objid);
        err_params_attr_objid = op_topo_child (err_params_comp_attr_objid, OPC_OBJTYPE_GENERIC, 0);
        op_ima_obj_attr_get (err_params_attr_objid, "Error Mode", &media_error_mode);
```

```
op_ima_obj_attr_get (err_params_attr_objid, "Bit Error Rate", &bit_error_rate);
op_ima_obj_attr_get (err_params_attr_objid, "Packet Error Rate", &packet_error_rate);
//*****************************************************************************************//

//*****************************************************************************************//
// James Song  songs@acm.org       May 2002
op_ima_obj_attr_get (my_objid, "Backoff Mode", &backoff_mode);
//*****************************************************************************************//

/* Based on physical charateristics settings set appropriate values to the variables. */
switch (phy_char_flag)
	{
	case WlanC_Frequency_Hopping:
		{
		/* Slot duration in terms of sec.*/
		slot_time = 5E-05;

		/* Short interframe gap in terms of sec.    */
		sifs_time = 2.8E-05;

		/* 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 sec.*/
		slot_time = 2E-05;

		/* Short interframe gap in terms of sec.    */
		sifs_time = 1E-05;

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

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

```
        break;
        }

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

        /* Short interframe gap in terms of sec.    */
        sifs_time = 1E-05;

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

        /* Maximum contention window size for selecting backoff slots.*/
        cw_max = 1023;
        break;
        }
//*****************************************************************************************//
// James Song  songs@acm.org       May 2002
// Objective:      Customize the WLAN's physical layer properties
case Customized:
        {
        /* Slot duration in terms of sec.*/
        op_ima_obj_attr_get (params_attr_objid, "Slot Time", &slot_time);

        /* Short interframe gap in terms of sec.    */
        op_ima_obj_attr_get (params_attr_objid, "Sifs Time", &sifs_time);

        /* Minimum contention window size for selecting backoff slots. */
        op_ima_obj_attr_get (params_attr_objid, "Min Contention Window", &cw_min);

        /* Maximum contention window size for selecting backoff slots.*/
        op_ima_obj_attr_get (params_attr_objid, "Max Contention Window", &cw_max);

        /* Slot duration in terms of sec.*/
        //slot_time = 5E-05;

        /* Short interframe gap in terms of sec.    */
        //sifs_time = 2.8E-05;
```

```
                /* Minimum contention window size for selecting backoff slots. */
                //cw_min = 15;

                /* 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;
                }
            }

        //*****************************************************************************************//
        // James Song  songs@acm.org      May 2002
        slots_utilization_handle          = op_stat_reg ("Wireless Lan.Slots Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
        possibility_transmission_handle    =    op_stat_reg    ("Wireless    Lan.Possibility    of
Transmission",OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
        //*****************************************************************************************//

        //*****************************************************************************************//
        // James Song  songs@acm.org      May 2002
        // Objective:     Initialize the err_data unit
        err_data.packets_received = 0;
        err_data.bits_received = 0;
        err_data.packets_damaged = 0;
        err_data.cur_packets = 0;
        err_data.cur_bits = 0;
        //*****************************************************************************************//

        //*****************************************************************************************//
        // James Song  songs@acm.org      May 2002
        // Objective:     Initialize the p_t data unit
        p_t.trans_flag = OPC_TRUE;
        p_t.vb_flag = OPC_FALSE;
```

```
        p_t.slot_utilization = 0;
        p_t.busy_slots = 0;
        p_t.init_backoff_slots = 0;
        p_t.p_trans = 1;
        p_t.attempts = 0;
        //*********************************************************************************************//

        //the rest part of wlan_mac_sv_init () is unchanged.
        // … …
}
```