



Computer Science

Panming Ye, Yong Zhou

Distributed Cross-layer Monitoring in Wireless Mesh Networks

Computer Science
D-level thesis

Date/Term: 09-06-11
Supervisor: Andreas Kessler
Examiner: Donald F. Ross
Serial Number: E2009:05

This report is submitted in partial fulfillment of the requirements for the Master's degree in Computer Science. All material in this report which is not my own work has been identified and no material is included for which a degree has previously been conferred.

Panming Ye, Yong Zhou

Approved, 2009-08-10

Opponent: **Marcel Castro**

Advisor: **Andreas Kessler**

Examiner: **Donald F. Ross**

Abstract

Wireless mesh networks has rapid development over the last few years. However, due to properties such as distributed infrastructure and interference, which strongly affect the performance of wireless mesh networks, developing technology has to face the challenge of architecture and protocol design issues. Traditional layered protocols do not function efficiently in multi-hop wireless environments. To get deeper understanding on interaction of the layered protocols and optimize the performance of wireless mesh network, more recent researches are focusing on cross-layer measurement schemes and cross-layer protocol design. The goal of this project is to implement a distributed monitoring mechanism for IEEE802.11 based wireless mesh networks. This module is event-based and has modular structure that makes it flexible to be extended. This project results a novel Cross-Layer Monitoring Module, CLMM, which is a prototype that monitors each layer of the nodes locally and dynamically, calculates the average values of the metrics, compares these values with thresholds and handles the cross-layer messages of each node. The CLMM also has a routing module structure that can be extended to distribute the metrics to its neighbors. Two simulations are analyzed to verify functionality of this module. The future work of the cross-layer monitoring module will also be discussed.

Acknowledgements

We would like to thank our supervisor, professor Andreas Kassler, for suggesting the topic of this thesis and supporting us patiently whenever we have questions. Without his help, this thesis would never have been finished. We are very grateful for all the support that we have received. We would also like to thank our tutor, Peter Dely, who helped us to fix all the problems we met and gave us many useful suggestions for the design, implementation and testing.

We would like to thank our examiner, Dr. Donald F.Ross, and our opponent, Marcel Castro, who spent time helping us to correct mistakes on writing and gave us a lot of comments and advice for thesis revising.

Contents

1 INTRODUCTION	1
1.1 WIRELESS MESH NETWORKS	1
1.2 PROBLEM DEFINITION AND MOTIVATION	3
1.3 PROJECT OBJECTIVES	5
1.4 THESIS ORGANIZATION	5
2 BACKGROUND AND RELATED WORK	6
2.1 IEEE 802.11S MESH NETWORKS	6
2.1.1 IEEE802.11s Architecture	6
2.1.2 Physical Layer in IEEE 802.11s	7
2.1.3 MAC Layer in IEEE802.11s	10
2.2 CROSS LAYER DESIGN AND CROSS-LAYER MONITORING	19
2.2.1 Layered Protocol and Cross-layer Protocol Design	19
2.2.2 Basic Architectures for Cross-layer Design	20
2.2.3 Cross-layer Monitoring	21
2.3 RELATED WORK	22
3 CROSS-LAYER MONITORING MODULE FOR WIRELESS MESH NETWORKS	26
3.1 INTRODUCTION	26
3.2 ARCHITECTURE OF THE CLMM	27
3.2.1 Monitor Module	28
3.2.2 Messages for the CLMM	31
3.2.3 Layer-Interface Module	33
3.2.4 Message Router	35
3.2.5 Storage Function	35
3.2.6 Trace Module	36
3.3 EXTENSIBILITY OF THE FRAMEWORK	36
3.3.1 Module Extensibility	36
3.3.2 Event Extensibility	37
3.3.3 Message Extensibility	37
3.3.4 Metrics Extensibility	38
3.3.5 Layer-interface Extensibility	39
3.4 METRICS	39
3.4.1 Introduction	39
3.4.2 Classification of Metrics	40
3.4.3 Metrics for Monitoring Module	43

3.4.4 Exponentially Weighted Moving Average (EWMA).....	49
4 IMPLEMENTATION.....	50
4.1 NS2.....	50
4.1.1 NS2 Architecture.....	50
4.2 NS-MIRACLE.....	52
4.2.1 NS-MIRACLE Architecture.....	52
4.3 IMPLEMENTATION OF THE CLMM.....	54
4.3.1 The MonitorPlugIn Module.....	56
4.3.2 The Storage Module.....	57
4.3.3 The Mac_Interface Module.....	58
4.3.4 The tmClMessage Module.....	59
4.3.5 Metrics module.....	61
4.3.6 Routing module.....	61
4.3.7 Ifhandler class.....	61
4.3.8 Interface class.....	61
4.4 OTCL SCRIPT.....	61
5 SIMULATION AND EVALUATION.....	64
5.1 SIMULATION SETTING.....	64
5.2 SIMULATION TOPOLOGY.....	65
5.3 SIMULATION RESULTS.....	66
5.3.1 Cross-layer messages.....	66
5.3.2 Results from Storage Module.....	67
5.3.3 Metrics.....	68
5.4 DISCUSSIONS.....	81
6 CONCLUSION AND FURTHER DISCUSSION.....	84
6.1 CONCLUSION.....	84
6.2 FUTURE WORK.....	85
6.2.3 Action Module.....	86
6.2.3 IP Routing Protocol.....	88
6.2.4 Summary.....	89
REFERENCES.....	90

List of Figures

<i>Figure 1-1</i> The Hybrid Wireless Mesh Networks [2].....	2
<i>Figure 2-1</i> IEEE802.11s Architecture [2]	7
<i>Figure 2-2</i> Clear Channel Assessment (CCA) [7]	10
<i>Figure 2-3</i> MAC Architecture [8].....	11
<i>Figure 2-4</i> The Basic Access Method [8].....	12
<i>Figure 2-5</i> Backoff Procedure [8]	13
<i>Figure 2-6</i> Interframe Space Relationships [8]	14
<i>Figure 2-7</i> An Implementation of EDCA [8].....	16
<i>Figure 2-8</i> Proposals of Cross-layer Architecture [11].....	21
<i>Figure 3-1</i> The Architecture of the Cross-Layer Monitoring Module (CLMM)	27
<i>Figure 3-2</i> The Structure of the Monitor Module.....	29
<i>Figure 3-3</i> Sequence Diagram of Query Message to Layer-interface Module	30
<i>Figure 3-4</i> Sequence Diagram of Query Message to Storage Module.....	31
<i>Figure 3-5</i> The Structure of Message for the CLMM.....	33
<i>Figure 3-6</i> The Structure of the Layer-Interface.....	34
<i>Figure 3-7</i> THE CALCULATION OF THE CHANNEL BUSY TIME AND THE CHANNEL IDLE TIME.....	44
<i>Figure 3-8</i> THE CALCULATION OF THE NUMBER OF THE FRAME RETRANSMISSIONS.....	47
<i>Figure 4-1</i> Architecture View of NS2 [26].....	51
<i>Figure 4-2</i> C++ AND OTCL: THE DUALITY [26].....	51
<i>Figure 4-3</i> Class Hierarchy (Partial) [26]	52
<i>Figure 4-4</i> An Example of NS-MARICLE Architecture [23]	53
<i>Figure 4-5</i> The Relationship of Classes of CLMM.....	55
<i>Figure 4-6</i> The structure of the MonitorPlugIn class	56
<i>Figure 4-7</i> The structure of the Storage Module.....	57
<i>Figure 4-8</i> The Structure of the Mac_interface Module	58
<i>Figure 4-9</i> The Structure of tmClMessage Module.....	60
<i>Figure 4-10</i> The Ifhandler Function	61
<i>Figure 5-1</i> Topology One.....	65
<i>Figure 5-2</i> Topology Two	66
<i>Figure 5-3</i> The Cross Layer Message Trace File	67
<i>Figure 5-4</i> A Layer Event Message.....	67
<i>Figure 5-5</i> A Queue Length Overflow Event Message.....	67
<i>Figure 5-6</i> The Output of Storage Module.....	68

<i>Figure 5-7 The Values of the Metrics (number of packets sent per second, number of packets received per second, busy time, number of one hop neighbors, number of busy period, queue length) in simulation1 using topology 1.....</i>	<i>71</i>
<i>Figure 5-8 The Values of the Metrics (number of packets sent per second, number of packets received per second, busytime, number of one hop neighbors, number of busyperiod, queue length) in simulation1 using topology 2.....</i>	<i>75</i>
<i>Figure 5-9 The Values of the Metrics (number of packets sent per second, number of packets received per second, busytime, number of one hop neighbors, number of busyperiod, queue length) in simulation2 using topology 1.....</i>	<i>78</i>
<i>Figure 5-10 The Values of The Metrics (number of packets sent per second, number of packets received per second, busytime, number of one hop neighbors, number of busyperiod, queue length) in simulation2 using topology 2.....</i>	<i>81</i>
<i>Figure 5-11 The Number of Packets sent per Second.....</i>	<i>82</i>
<i>Figure 5-12 The Number of Packets Received per Second.....</i>	<i>83</i>
<i>Figure 6-1 The Initialization Function.....</i>	<i>85</i>
<i>Figure 6-2 The structure of configuration file and the Initialization Function.....</i>	<i>86</i>
<i>Figure 6-3 The Structure of the Action Module.....</i>	<i>87</i>

List of Tables

<i>Table 2-1 The Standardization of the Physical Layers [8]</i>	8
<i>Table 2-2 UP-to-AC Mappings [8]</i>	15
<i>Table 3-1 The message types</i>	32
<i>Table 3-2 The Sub-types of the Messages for the CLMM</i>	33
<i>Table 3-3 The data types for the database</i>	36
<i>Table 3-4 Metrics that are Implemented for CLMM in Phase1</i>	40
<i>Table 3-5 Metrics that to be Implemented for CLMM in Phase2</i>	42
<i>Table 3-6 Metrics that to be Implemented for CLMM in Phase3</i>	42
<i>Table 4-1 Variables of the Metrics</i>	63
<i>Table 5-1 Simulation Setting</i>	65
<i>Table 5-2 Setting in a Test Scenario</i>	82

1 Introduction

1.1 Wireless Mesh Networks

In recent years, with the rapid development of various wireless networks, the technology of wireless mesh networks (WMNs) has emerged. In wireless mesh networks, a collection of wireless nodes communicate with peers in one or multiple hops. Therefore, it can extend the area of wireless broadband coverage without wiring network, or it can be used for temporary installation or extension to LAN/WLANs without wired connection. Several wireless network technologies such as Wi-Fi (based on IEEE802.11), WiMAX (based on IEEE802.16), can be deployed as mesh networks to provide a wireless distribution system to access other networks (i.e. the Internet). The cost of deploying a large scale wireless mesh network service can be reduced dramatically because of the lack of a wired infrastructure. Therefore, the wireless mesh network technology provides an option that can provide low-cost broadband access services for the “last-mile” access networks.

WMNs consist of two types of nodes: mesh routers and mesh clients [1]. As shown in Figure 1-1, wireless mesh routers have minimum mobile ability form the wireless backbone [2]. Mesh routers provide additional multi-hop routing functions for mesh networking. Through multi-hop communications, a wireless mesh router can reach the same coverage with much lower transmission power than a conventional wireless router. A mesh client node does not have gateway or bridge functions. However, the mesh client node has basic mesh routing functions and can also work as a router. Mesh clients access the network through mesh routers or other mesh clients. The hardware platform and software of the mesh clients are much simpler than mesh routers [1].

Based on the functionality of the nodes, the architecture of WMNs can be classified into three main groups: Infrastructure/Backbone WMNs, Client WMNs and Hybrid WMNs [1]. Figure 1-1 shows the examples of these three groups. Infrastructure/Backbone WMNs has an infrastructure formed by wireless mesh routers. And infrastructure/Backbone WMNs provide backbone for clients and enable integration of WMNs with existing wireless networks. Client WMNs do not have an infrastructure. Client WMNs provide peer-to-peer networks among client devices. Client nodes comprise the actual network to perform routing and configuration functionalities as well as providing end user applications to customers. Hybrid WMNs are the

combination of infrastructure and client meshing. The infrastructure provides connectivity to other networks (i.e. the Internet, Wi-Fi or WiMAX).

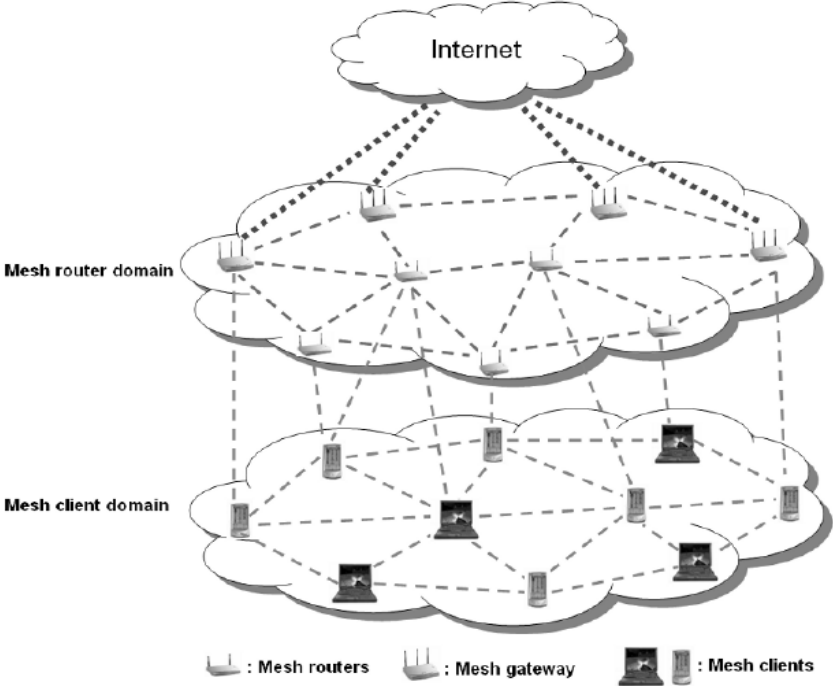


Figure 1-1The Hybrid Wireless Mesh Networks [2]

In a wireless mesh network, each entity operates independently. The main characters of the wireless mesh network are summarized [1] as follow:

- First, a wireless mesh network is a multi-hop wireless network which supports multiple hops forwarding. When new nodes join the wireless mesh network, the coverage, reliability and connectivity are enhanced by extending the coverage range without reducing the channel capacity. However, multi-hop networking also increases link failures, which can reduce throughput sharply.
- Second, wireless mesh networks support ad hoc networking, and capability of self-forming, self-healing, and self-organization. The nodes in the mesh network automatically setup and maintain network connectivity. Due to these properties, network performance is enhanced by increasing reliability and robustness. And it is easy to be deployed and extended in large area.
- In wireless mesh network, the mobility is depended on the type of mesh nodes. The mesh routers usually have minimal mobility, while mesh clients can be stationary or mobile nodes.

- Power consumption is dependent on the different type of nodes. Mesh routers which have minimum mobility do not have strict constraints on power consumption. However, in some cases (i.e. in wireless sensor network), mesh clients may require power efficient protocols. Therefore the MAC or routing protocols designed for mesh routers may not be appropriate for mesh clients.

Although wireless mesh networks have several advantages, there exist several challenges. Advanced wireless radio technologies such as cognitive radio, MIMO and smart antennas are complicated and the cost of these technologies is still too high for commercial applications [1]. Interoperability and integration of heterogeneous networks should be improved. The protocols such as MAC protocol and routing protocols should support the mobile ability, multiple hops, robustness and scalability of WMNs efficiently. Also, the network performance should meet QoS requirements to support the real-time multimedia applications. All above challenges bring new research issues to protocol designs of wireless mesh networks. One of the key issues is cross-layer design, which optimizes multiple layers, primarily the physical (PHY), medium access control (MAC) and network layers to enhance the capacity of wireless mesh networks efficiently.

1.2 Problem Definition and Motivation

Although a layered protocol has the flexibility in upgrading certain layers, easy debugging, and low complexity, the traditional layered protocols can not function efficiently in wireless mesh networks. Due to the properties of dynamic nature, multi-hop communication and no central instance, wireless mesh networks are significantly different from the traditional wired networks. For example, TCP protocol does not work efficiently in wireless mesh networks due to the properties of multiples hops and dynamic nature [3]. To improve the performance of the transport layer, [3] suggests a cross-layer approach that combines MAC layer and TCP layer together. In another example, [4] shows that the routing layer in wireless mesh networks needs to work interactively with the MAC layer in order to maximize its performance. Because the possibility of performance advantage in wireless mesh network, cross-layer design issues are capturing the interest of the research community recently.

Not only the cross-layer protocol design is important to optimize the performance of the network layers, but also the cross-layer monitoring is useful for observation of the network behaviors. Cross-layer monitoring not only observes the network behaviors, interactivities of the protocols and mechanisms used in wireless mesh networks, but also helps to improve the

interoperation of protocols and mechanisms. For example, in IEEE802.11s mesh networks, there is a local congestion control mechanism that enables MAC and IP layer work together to reduce the congestion [5]. With cross-layer monitoring, nodes can collect the values of metrics and distributes the information to the layers. Furthermore, each node can distribute its information to its neighbors. Thus layers and nodes can get the global knowledge of operational properties of the mesh network, and improve the network performance by adjusting parameters such as sending rate based on the monitored information.

To gain a better understanding of the interaction of these layered protocols and improve the performance of wireless mesh network, we motive a cross-layer monitoring module for wireless mesh networks. This module is a distributed and event driven cross-layer monitor module. It not only observes the operational characteristics of layers, but also has the functions to handle and process the values of monitored metrics for the layers. Furthermore, it has the function to make one layer messages available for other layers or other nodes. The advantages of the cross-layer monitoring module for wireless mesh network are twofold:

- First, cross-layer monitoring can help us examine each layer in a deep insight view. Collecting information from each layer can help each layer know and understand the status of other layers. Having knowledge of the every layer can help us to find out how these layers interact with each other. We can also set parameters of other layers to find out how one parameter of one layer affects other parameters of other layers during the communication.
- Second, since we know how each layer interacts with each other, we can optimize the network performance by adjusting their parameters accordingly. To gain the best network performance, each layer should work together, and a collaboration mechanism is needed for this proposal. For example, in IEEE 802.11s draft standard, there is a local congestion control mechanism which needs MAC protocol and routing protocol work together to improve the performance. Therefore the cross-layer monitoring module will be helpful for this mechanism.

This module should be configurable, adaptive and extendible. And the implementation will be done in NS2 with NS-MIRACLE extension. According to that, metrics of the monitoring module should be selected carefully. First, these metrics can be used to improve the network performance efficiently. Second, they can be realized in NS2 with NS-MIRACLE. Furthermore, metrics, with the different monitoring goals, can be added to the framework of the module.

Based on the collected metrics, layers and nodes can take appropriate actions and thus improve performance. The monitoring module has a function that can compare the monitored values of the metrics with thresholds. When the monitored values of a metric (i.e. MAC busy time) reach the thresholds, the layer sends out a message to the monitoring module and the monitoring module handles the message accordingly.

This Monitoring module has a structure that can be extended to distribute a set of metrics to its neighbors. The distribution is done by IP routing protocol. The information coming from its neighbors is stored in the routing table. The monitoring module reads this information and processes it for network performance optimization.

Finding those heuristics and performing the actions are not part of the project. However, the architecture of the monitoring module is flexible so that it can be easily extended to a monitoring/control agent, which triggers such actions for optimizing network performance.

1.3 Project Objectives

The goal of this monitoring module design is twofold: first, we learn how to design and implement a cross-layer monitoring module for wireless mesh networks. We will learn not only how to design the architecture of the module, but also how to implement metrics of layers for the monitoring module and how to make the module process and distribute the messages. Second, we monitor these metrics and study interactions of layers and nodes. A reaction mechanism is considered in the monitoring module. This reaction function makes it possible to control each node's network behaviors based on the monitored metrics. Furthermore, a distributing mechanism is motivated to distribute information to neighbor's nodes, which can be used for performance optimization in wireless mesh networks.

1.4 Thesis Organization

The rest of this thesis is structured as follow: Chapter2 introduces the background of IEEE 802.11 based mesh networks and related works of cross-layer protocol design in wireless networks. Chapter3 presents the architecture, functions of the cross-layer monitoring module and the metrics for the module. Chapter4 introduces the details of implementation of the monitoring module. We will also introduce NS2 and NS-MIRACLE briefly in this chapter. In Chapter5, we evaluate the monitoring module in NS2 with NS-MIRACLE extension and describe the simulation in detail. Chapter6 summaries the project and discuss the future work of the project.

2 Background and Related Work

This chapter introduces background information about IEEE 802.11s Wireless Mesh Networks. First, we describe the architecture of IEEE 802.11s mesh network in section 2.1. Also, we introduce basic components of physical layer, MAC layer, routing protocols and challenges of IEEE802.11s in this section. Then we will present cross-layer design issues in section 2.2-2.5.

2.1 IEEE 802.11s Mesh Networks

Wireless mesh networking based on IEEE 802.11 wireless local area network (WLAN) has been developed in the last few years. In 2004, the increasing demand for mesh networks led to the formation of a task group (TGs) to define the Extended Service Set (ESS) Mesh Networking Standard. IEEE 802.11s provides frame forwarding over multiple hops and path selection at data link layer. It provides a transparent support to any higher layer protocols [6] and seamlessly integrates in the Institute of Electronics and Electrical Engineering (IEEE) 802 set of standards.

2.1.1 IEEE802.11s Architecture

[4], [5], [6], [7] present the architecture of IEEE802.11s mesh network. The network architecture is shown in Figure 2-1. An IEEE802.11s mesh network consists of three types of nodes: mesh points, mesh access points, and mesh portals. The basic element in IEEE802.11s is the mesh point (MP). Mesh point supports the mesh services of control, management, and operation of the mesh. A mesh point (MP) can exchange frames with other nodes through multiple wireless hops. MAC Service Data Unit (MSDU) [8], which is the information that is delivered as a unit between MAC service access points (SAPs), is exchanged between mesh points. A STA is a device that contains an IEEE 802.11-conformant medium access control (MAC) and physical layer (PHY) interface to the wireless medium [8]. A mesh access point (MAP) is an MP but can also work as an access point that gives stations access to network services. The STAs do not participate in the WLAN mesh, but they can associate with the mesh access points (MAPs) to connect to the mesh networks. Each mesh point can operate like a station (STA) or forward data frames for other nodes. A Mesh Portal (MPP) is an MP that interconnects IEEE802.11 wireless LANs and other networks.

A mesh link is built by two mesh points who can directly communicate with each other via the wireless medium. A pair of nodes that share a link are neighbors. Mesh BSSs are connected via wireless mesh networking.

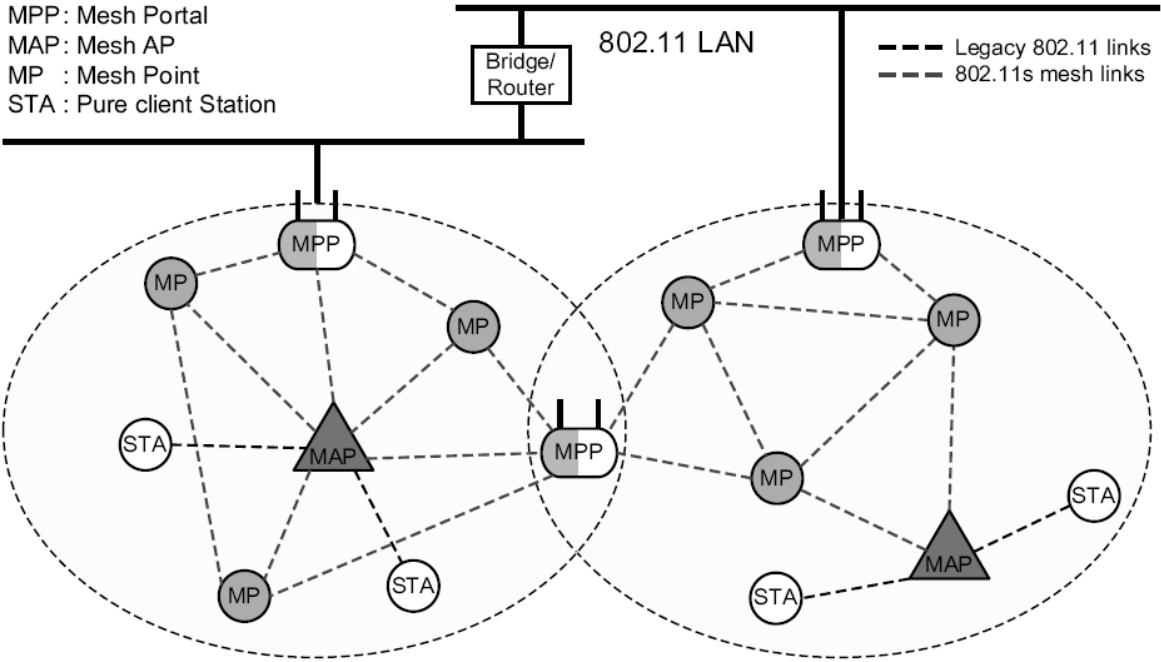


Figure 2-1 IEEE802.11s Architecture [2]

2.1.2 Physical Layer in IEEE 802.11s

2.1.2.1 Physical Layer in IEEE 802.11

The PHY layer in IEEE 802.11 consists of two sub-layers: the physical layer convergence procedure (PLCP) sub-layer and the physical medium dependent (PMD) sub-layer [8]. The PLCP is between the frames of MAC and radio transmissions in the air. It adds its own header and provides a mechanism for transferring MPDUs between two or more STAs over the PMD sub-layer [8]. PMD is a method of transmitting and receiving data via a wireless medium using the antenna. Data transmission over the wireless media is controlled by PMD. It transforms the PLCP protocol data unit (PPDU) into RF signal by using carrier modulation. Three physical layers are standardized in the original revision of 802.11: Frequency-hopping spread-spectrum (FHSS) radio PHY, Direct-sequence spread-spectrum (DSSS) radio PHY and Infrared light (IR) PHY. However, the infrared physical layer has never been implemented in a commercial product. From 1997 to 2007, four more physical layers based

on radio technology have been developed. The standardization of the physical layers is shown in Table2-1.

		Frequency Band(GHz)	Max Data rate(Mbps)	Modulation	Slot time(μ s)	SIFS time(μ s)	Contention window size	Preamble duration(μ s)	PLCP header duration(μ s)	Maximum MAC frame	Minimum sensitivity(dBm)
1	FHSS	2.4	2	GFSK	50	28	15-1023	96	32	4095	-80
2	DSSS	2.4	2	QPSK	20	10	31-1023	144	48	4-8191	-80
3	IR										
4	OFDM	5	54	OFDM	9	16	15-1023	20	4	4095	-62
5	HR/DSSS	2.4	11	Barker Code / CCK	20	10	31-1023	144	48	4095	-76
6	ERP	2.4	54	Barker Code / CCK / OFDM	20 or 9	10	15 or 31 to 1,023	20		4095	
7	MIMO-OFDM		100+		9	16	15-1023	16	4	8191	-64

Table 2-1 The Standardization of the Physical Layers [8]

Basic Spread Spectrum Techniques

As shown in Table 2-1, there are six basic spread spectrum techniques used in IEEE802.11: Frequency Hopping Spread Spectrum (FHSS), Direct Sequence Spread Spectrum (DSSS), Orthogonal Frequency Division Multiplexing (OFDM), High-Rate Direct Sequence (HR/DSSS), Extended Rate PHY (ERP) and multiple-input multiple-output (MIMO) OFDM.

- FHSS uses the 2.4GHz frequency band as the RF transmission media. It uses Frequency-hopping systems to jump from one frequency to another in a random pattern, transmitting a short burst at each sub-channel. FHSS uses Gaussian frequency shift keying (GFSK) modulation. The number of non-overlapping channels is 79 for the United States and Europe. The channel center frequency is defined in sequential 1.0 MHz steps beginning with the first channel.
- DSSS also works in 2.4GHz band. DSSS spreads the power out over a wider frequency band using mathematical coding functions. It provides a WLAN with both a 1 Mb/s and a 2 Mb/s data rates. DSSS uses two baseband modulations: the differential binary phase shift keying (DBPSK) and the differential quadrature phase shift keying (DQPSK). The basic access rate is based on 1 Mb/s DBPSK modulation. The enhanced access rate is based on 2 Mb/s DQPSK. The initial 2Mbps PHY is specification in [8]. The number of operating channels in DSSS is

14. The width of each channel is 22MHz. The adjacent channels overlap one another partially, with three of the 14 being completely non-overlapping.
- OFDM is operated in the 5 GHz band. It uses carrier multiplexing technique to divide an available channel into several sub-channels. Each sub-channel is used to transmit a fraction of the user data stream at the same time. By making these carriers orthogonal to each other they do not interfere. All the sub-channels are then multiplexed into a combined channel to increase the network throughput. The OFDM system provides the communication capabilities up to 54 Mb/s.
 - High-Rate Direct Sequence (HR/DSSS) is an extension of the PHY for the DSSS system. HR/DSSS works in 2.4GHz band and provides 5.5 Mb/s and 11 Mb/s data rates. HR/DSSS defines two PLCP preambles: the long PLCP preamble and the short PLCP preamble. To provide the higher rates, Complementary Code Keying (CCK) is used as the modulation scheme. CCK is based on sophisticated mathematical transforms. It consists of a set of 64 8-bit code words. It allows the use of a few 8-bit sequences to encode 4 or even 8 bits per code word, for a data throughput of 5.5 Mbps or 11 Mbps.
 - Extended Rate PHY (ERP) is operated in 2.4GHz band. It uses different modulations for different data rates. The major modulation is ERP-OFDM. It supports the same speeds as IEEE802.11a: 6, 9, 12, 18, 24, 36, 48, and 54 Mbps. Speeds of 6, 12, and 24 Mbps are mandatory. Other modulations are: ERP-DSSS/CCK for the data rate of 1, 2 Mb/s and 5.5Mb/s, ERP-PBCC for the data rate of 22Mb/s and 33Mb/s and DSSS-OFDM. DSSS-OFDM is a hybrid scheme, which encodes packet data using the DSSS headers, and OFDM encoding of the payload.
 - MIMO-OFDM uses 2.4GHz and 5GHz unlicensed bands. It supports 20MHz and 40MHz channel. MIMO uses two or more antennas to send or receive one or more data streams at the same time. Using multiple antennas, MIMO-OFDM combines OFDM and MIMO techniques. This not only increases the data transmission rate and enhances the transmission reliability, but also gains longer transmission range. Independent OFDM modulated data are transmitted from multiple antennas simultaneously. At the receiver, data are OFDM demodulated first. Then MIMO decoding on each of the sub-channels extracts the data from all the transmit antennas on all the sub-channels.

Clear Channel Assessment

In IEEE 802.11, the physical layer supports a clear channel assessment (CCA) function to indicate to the MAC when a signal is detected. To implement the CSMA/CA in IEEE 802.11, the PLCP has a function to determine whether the wireless medium is busy. The structure of CCA is shown in Figure 2-2 [7]. The Physical Carrier Sense (P-CS), which is the part of the Physical Layer (PHY)'s Clear Channel Assessment (CCA), is used to determine the state of the medium in physical layer. With P-CS, each node senses energy in the air. If the energy is over the thresholds, it indicates a busy medium. Otherwise, it shall be considered idle. The threshold value depends on the IEEE 802.11 PHY layer.

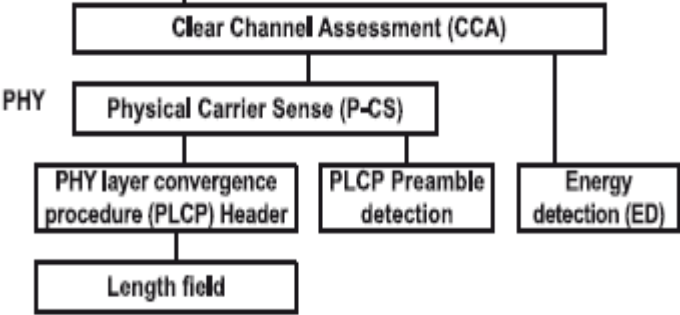


Figure 2-2 Clear Channel Assessment (CCA) [7]

2.1.2.2 Physical layer in IEEE 802.11s

IEEE 802.11s physical layer is set up based on IEEE802.11 standard. Although there are advanced wireless radio technologies such as smart antenna, multi-input/multi-output (MIMO) and cognitive radios that have been proposed in wireless mesh network in recent years, these new technologies are complicated and increase the hardware costs. The network cost has become a challenging problem in IEEE 802.11s mesh networks. All these radio technologies require a novel design in higher layer protocols, especially MAC and routing protocols. To date, MIMO is already the key technologies for IEEE 802.11n, but it is still optional in IEEE802.11s draft.

2.1.3 MAC Layer in IEEE802.11s

Due to multi-hop forwarding, the transmission of one hop may affect the next hop or any links in the neighbors. The available resources must be efficiently allocated for the network to serve a large coverage area. Functionalities need to be enhanced to improve the performance of the wireless mesh networks. In this section, we first introduce IEEE802.11 MAC layer, then we introduce the basic components of MAC layer in IEEE802.11s.

2.1.3.1 MAC Layer in IEEE 802.11

The architecture of the MAC layer is shown in Figure 2-3 [8]. In IEEE 802.11, accessing wireless medium is controlled by three basic coordination functions: the point coordination function (PCF), distributed coordination function (DCF) and the hybrid coordination function (HCF). The PCF and HCF are provided via the services of the DCF. The PCF provides contention-free services and is restricted to infrastructure networks. However, the PCF is optional in all STAs and not implemented in real wireless networks. HCF is only present in a QoS STA. In a QoS STA implementation, both DCF and HCF are present.

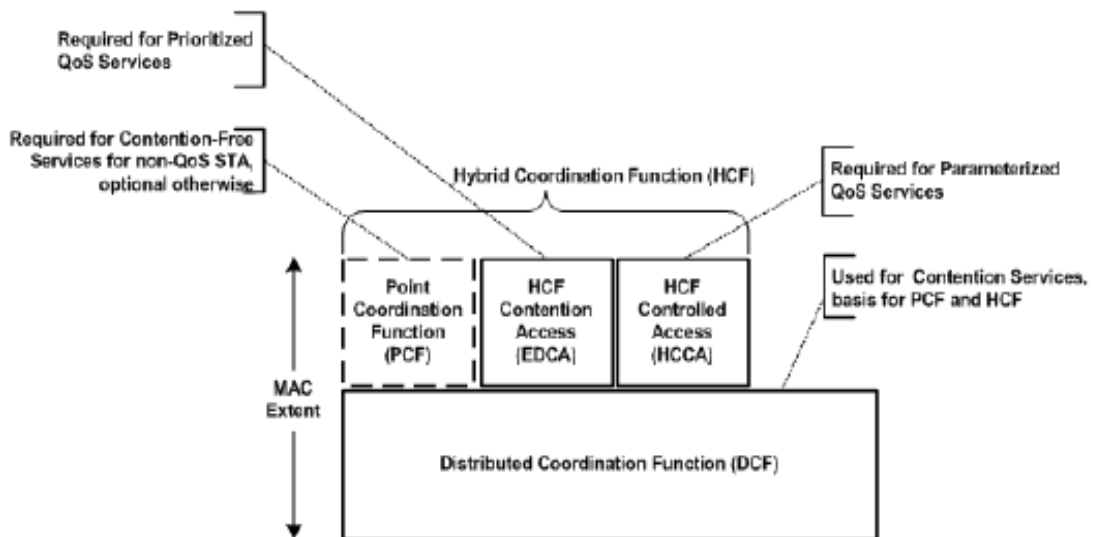


Figure 2-3 MAC Architecture [8]

IEEE802.11 DCF is the basic medium access protocol which uses Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) and random backoff mechanism to reduce contention in the wireless medium. As shown in Figure 2-4, the basic access method is summarized by [9] as below:

- Before a node transmits a frame, it uses carrier sensing to determine if the medium is available.
- If the channel is idle for a Distributed Interframe Space (DIFS), the station transmits the frame to the destination.
- If the channel is busy, the station waits for a specific period of time called the backoff interval, and then tries to sense the medium again.
- The destination sends an acknowledgment to the source if it successfully receives the frame.

- If the source does not receive an acknowledgment within a specific period of time, it tries to send the frame again.
- Retransmission starts if the channel is idle for an Extended Interframe Space (EIFS).
- When a unicast frame is received, an acknowledgement shall be transmitted by the destination. Broadcast frames do not require the destination to send an acknowledgment of reception.

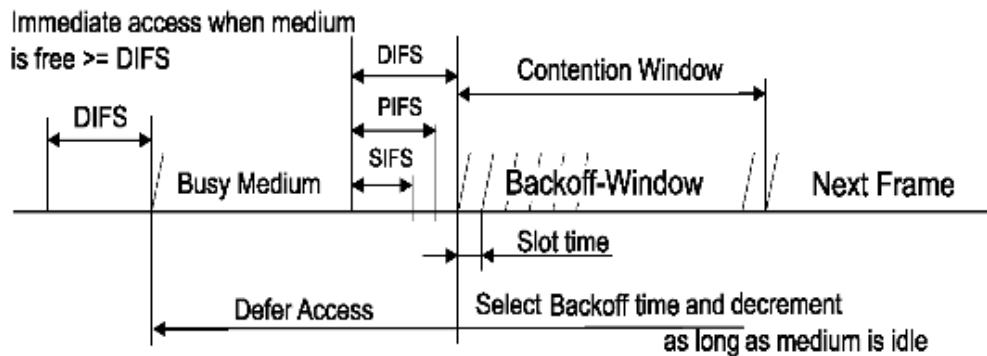


Figure 2-4 The Basic Access Method [8]

In IEEE 802.11, two types of carrier-sensing functions are used: the physical carrier-sensing (P-CS) and virtual carrier-sensing functions (V-CS). The medium is considered as busy if either carrier-sensing function indicates that the channel is busy. The physical carrier-sensing (P-CS) which implemented in physical layer is described in section 2.1.2. Virtual Carrier Sensing (V-CS) is implemented in MAC layer. It is provided by the Network Allocation Vector (NAV). IEEE802.11 frames carry a duration field in the MAC header. The duration fields can be used to reserve the medium for fixed time period. The NAV is a count-down timer that indicates the amount of time the medium will be reserved. As the node starts to initiate a frame, it sets the NAV to the time for which it expects to use the medium. All nodes monitor the wireless medium and get reservation information from any frame they could decode. Then the nodes set their NAV to the according value. To prevent collisions caused by hidden nodes, IEEE802.11 uses Collision Avoidance (CA) scheme.

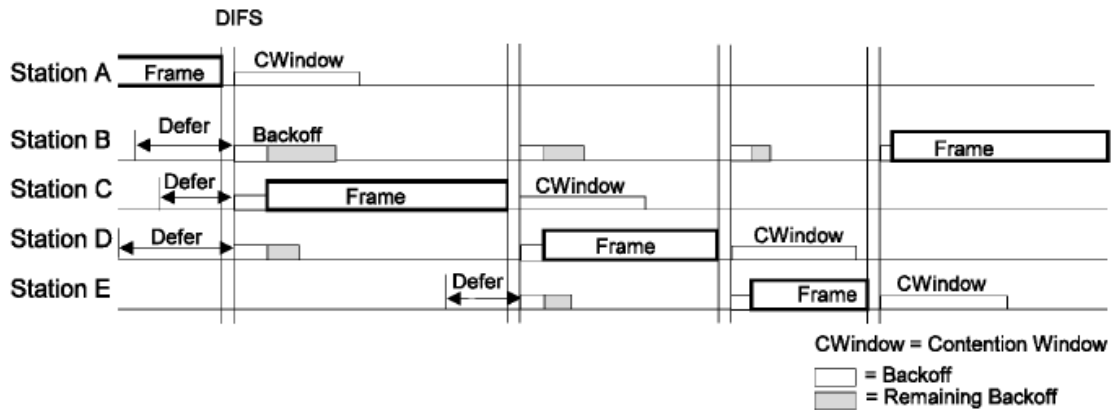


Figure 2-5 Backoff Procedure [8]

The basic backoff procedure is shown as Figure 2-5. First, before nodes initiate a frame transmission, they sense the wireless medium for a duration DIFS. If the wireless medium is idle, nodes perform a random backoff time. A node may send a frame if it senses the medium is idle after a backoff time. The backoff time works like a counter. If either of the P-CS or V-CS indicates busy medium conditions, the nodes freeze the counter and wait until the wireless medium is idle again. The NAV counter is decremented by one whenever the wireless medium is idle. Before the NAV is zero, the virtual carrier-sensing function indicates that the medium is busy; when the NAV reaches 0, the virtual carrier-sensing function indicates that the medium is idle. The frame is sent out right after the NAV reaches zero.

The backoff timer value is calculated as:

$$\text{Backoff_time} = \text{random}() * \text{SlotTime} \quad \text{Equation 2-1}$$

Where the random () is pseudo-random integer drawn from a uniform distribution over [0, CW], where CW is contention window and CW is in [CW_min, CW_max].

Collision avoidance is done using timing intervals. To provide different priority levels for the different types of traffic, IEEE802.11 provides five interframe spaces: SIFS, PIFS, DIFS, EIFS and AIFS. Figure 2-6 shows the relationship of IFS. The short interframe space (SIFS) is shortest and used for the highest-priority transmissions, such as RTS/CTS frames and positive acknowledgments. The DCF interframe space (DIFS) is the minimum medium idle time for contention-based services. Nodes may access to the medium immediately if it has been free for a DIFS. The Extended interframe space (EIFS) is not a fixed interval and it is used only when there is an error in frame transmission. AIFS is used by QoS STAs to transmit frames.

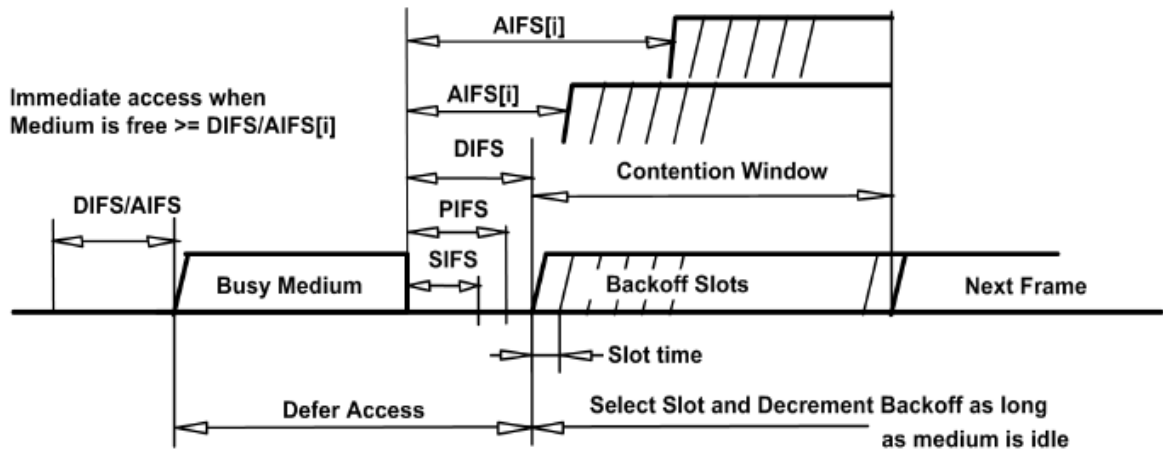


Figure 2-6 Interframe Space Relationships [8]

By using the NAV, nodes can ensure that atomic operations are not interrupted. When multiple nodes contend for wireless medium and go into random backoff, the node selecting the smallest backoff time using the random function will win the contention. Along with the Collision Avoidance scheme, it also uses immediate positive acknowledgment (ACK frame) for each frame received. All the unicast frames sent by a node to a receiver must be acknowledged. If no ACK is received, the CW Size is doubled by the sender and the sender retransmits frame. In some cases, the DCF may also use the CTS/RTS clearing technique to avoid collisions. However, CTS/RTS mechanism is never used in real wireless mesh networks [7].

2.1.3.2 MAC Layer in IEEE802.11s

A. EDCA

The basic coordination function (CF) mechanism of 802.11s MAC is the enhanced distributed channel access (EDCA) [5]. The EDCA mechanism provides differentiated, distributed access to the wireless medium for STAs using eight different user priorities (UPs) [8]. As shown in Table 2-2, these UPs are mapped to four access categories (ACs): Best effort, Background, Video and Voice. These ACs provide support for the delivery of traffic with UPs at the STAs.


Priority	UP (Same as 802.1D user priority)	802.1D designation	AC	Designation (informative)
Lowest  Highest	1	BK	AC_BK	Background
	2	—	AC_BK	Background
	0	BE	AC_BE	Best Effort
	3	EE	AC_BE	Best Effort
	4	CL	AC_VI	Video
	5	VI	AC_VI	Video
	6	VO	AC_VO	Voice
	7	NC	AC_VO	Voice

Table 2-2 UP-to-AC Mappings [8]

Figure 2-7 shows an implementation of EDCA. Each AC has its own frame queue and backoff entity using different parameter set for medium access. EDCA uses arbitration Interframe Space (AIFS) instead of DIFS (shown in Figure 2-6). The duration of AIFS is calculated as:

$$AIFS [AC] = AIFSN [AC] \times aSlotTime + aSIFSTime \quad [8] \quad \text{Equation 2-2}$$

Arbitration IFS Number (AIFSN), CWmin and CWmax depend on the AC. The AIFS varies with the priority of the packet. Compared to DCF, EDCA changes the fairness principle. When stations perform EDCA, they contend for Transmission Opportunities (TXOPs) which are depended on the AC. The higher priority uses a smaller AIFS [AC] and CW [AC] value, which means the higher priority queue gets more chance to use the channel. The station may send frames if it does not exceed the TXOP limit. Furthermore, EDCA supports Block ACK [8] and provides equal transmission duration to all stations. It operates more efficiently than DCF in one hop wireless networks. However, because its prioritization mechanism does not perform well in a multi-hop mesh environment, EDCA does not work well for mesh networks [4], [7], [10].

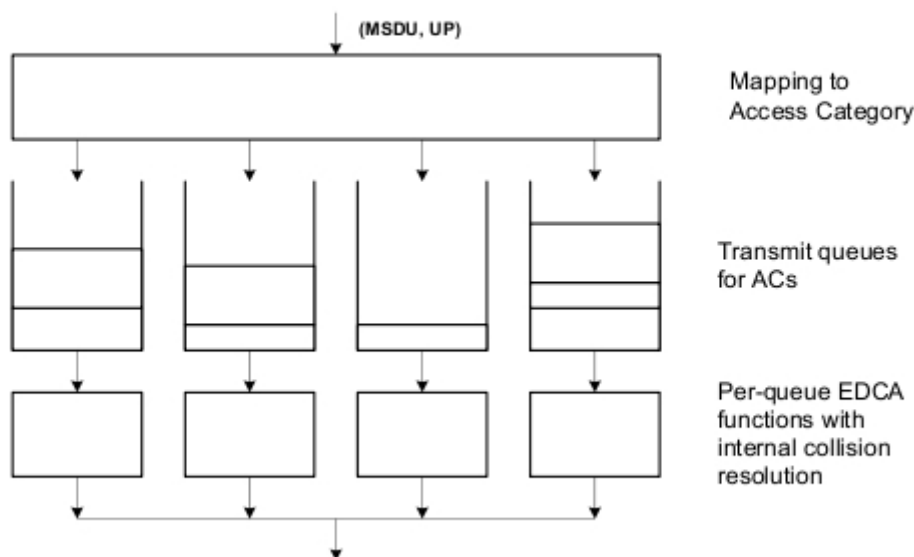


Figure 2-7 An Implementation of EDCA [8]

B. Synchronization

[4], [5], [6] described Synchronization in 802.11s, which is an optional feature for MPs. IEEE 802.11s defines beaconing procedures for unsynchronizing and synchronizing MPs. Two mechanisms are used to avoid beacon collisions. First, not all mesh points timing system function (TSF) should be synchronized. A TBTT (Target Beacon Transmission Time) offset is needed for mesh point's synchronization. The TSF is calculated by using this TBTT offset and the beacon timestamp. The value of the TSF can be different. The second mechanism is mesh beacon collision avoidance (MBCA) mechanism. IEEE 802.11s adds information elements (IEs) to the standard beacon frame and probes response frame to provide TBTT information.

With synchronization, each MP maintains a common Mesh TSF time by collecting beacon timing information with time stamp and offset received from neighbors. Furthermore, MPs use beacon frames to detect each other, to maintain connectivity and to synchronize their local clocks.

C. Extensions to the Medium Access Control in 802.11s

The 802.11s draft standard defines an optional hop-by-hop congestion control mechanism [5]. The whole process of congestion control mechanism has three modules, which are Local Congestion Monitoring, Congestion Control Signaling and Local Rate Control. The basic idea

of the mechanism is that each MP observes the level of congestion based on the transmitting rate and the receiving rate of packets that need to be forwarded. When the traffic increases and the MP can not forward and source data upstream as fast as the incoming rate, congestion occurs. Queue size can also be used to monitor congestion. Each MP regulates incoming and outgoing data to minimize the transit queue size. With sufficient queue size, a notification of congestion is sent to its one-hop neighbors. These neighbors adjust their sending rate accordingly at which they are sending to the congested MP.

To perform local congestion control mechanism in IEEE 802.11s mesh networks, messages are necessary to be monitored and exchanged within different layers. Traditional layered protocols are not efficiently supporting the functionality. Therefore, cross-layer protocol design should be considered for the mechanism.

D. Power management

The goal of power management for WMNs is to improve the power saving, network connectivity and throughput. In wireless mesh networks, dependence of power-consumption constraints on the type of mesh nodes. Power efficiency is the major concern for the light weight MPs and mesh clients. MAPs are required to be active continuously. However other kinds of nodes such as MPs or mesh clients need a power save mechanism for saving power source. When the MPs do not forward traffic for other nodes, they will work in power save mode. At the same time, as various transmissions power level not only causes different level of interference, but also causes various sending rate, power management is closely coupled with MAC protocols. Since the connectivity affects performance of a routing protocol, power management is also crucial for the network layer [2].

In IEEE 802.11s mesh network, the power management is done by the announcement traffic indication message (ATIM) mechanism. The MPs work in either doze state or wake state. The Mesh BSS has a common Mesh delivery traffic indication message (DTIM) interval. The power saving MPs periodically wake up during the ATIM window to receive or send control messages including beacons. The ATIM window repeats every one DTIM interval. During the duration, peer MPs send buffered unicast frames or request the MP to remain in wake mode for further frame delivery. An MP may also wake up in a scheduled time period negotiated with other MPs. In the power save mode, packets in an MP need to be buffered and wait to be sent during the wake state [4].

2.1.3.3 Link Management

Once a new mesh node powers up, it needs to establish peer links with its neighbors at first. Mesh Points (MPs) use passive or active scanning to set up peer links [6]. By using passive scanning, MPs listen for beacon frames. Active scanning includes transmission of probe request frames. IEEE 802.11s uses a new mesh ID to identify a mesh network. The mesh ID is attached in beacon and it probes response frames as a new information element (IE). Once a Mesh Link (ML) has been established, it is also necessary to perform a measure of link quality for each peer link. The link quality information of each peer link can also be used as one of the routing metrics for the routing protocol. The default airtime metric of peer MPs is calculated as:

$$ca = (Oca + Op + Bt/r) / (1 - ept) \quad \text{Equation 2-3}$$

Where Oca is Channel access overhead, Op is Protocol overhead, Bt is the Number of bits in a test frame, r is MCS bit rate and epf is Frame error rate for the test frame [5].

2.1.3.4 Path selection and routing

In IEEE802.11s, path selection and forwarding is performed at MAC layer. The IEEE 802.11s framework allows multiple path selection protocols being implemented in a mesh point. However, only one protocol is active in a Mesh BSS each time. Mesh Points use the WLAN Mesh Capability IE to indicate which protocol is in use. The default routing protocol in IEEE 802.11s is called hybrid wireless mesh protocol (HWMP) [5]. HWMP is a hybrid routing protocol of on-demand routing and proactive tree-based routing. The on-demand routing protocol addresses the mobility of the mesh nodes. It is similar to Ad-hoc On-demand Distance Vector (AODV), which uses a simple hop count routing metric [5]. Mesh points broadcast beacon frames periodically. The beacon frames carry path selection and topology information. MPs can use a Route Request (RREQ) and Route Reply (RREP) mechanism to discover link metric information from source to destination. For example, the source mesh point A broadcasts a RREQ first before it sends data to a destination mesh point B. The intermediate nodes store the address of the MP they received the RREQ from and re-broadcast the RREQ. When B receives RREQ, it replies with a unicast RREP addressed to A. The RREP follows a reverse path to A and the path is established.

The proactive tree-based routing protocol configures a root MP in the mesh network. It builds and maintains a distance vector tree for other nodes to avoid routing overhead for routing path discovery and recovery. All MPs maintain a path to the root MP proactively. The tree can be set up in two ways:

- The Portal broadcasts Proactive RREQ (PRREQ) frames, where every MP replies with a Gratuitous RREP (GRREP).
- The Portal broadcasts Portal Announcement (PANN) frames, leaving each MP the possibility to set up the path whenever it needs in an on-demand fashion.

The on-demand routing and tree-based routing schemes can run at the same time. Airtime cost is the mandatory routing metric for the routing protocol that measures the quality of links [4]. Other types of metrics such as QoS parameters, traffic load, power consumption, can also be used and only one metric can be used in the same mesh at a time.

2.2 Cross Layer Design and Cross-layer Monitoring

2.2.1 Layered Protocol and Cross-layer Protocol Design

In traditional layered protocol model, each layer is independent and has no knowledge of the other layer. Each layer provides a specific service. Each service at a layer is realized by designing protocol for the layer. Every layer manages its own variables, and its variables are of no concern to other layers. The communication between adjacent layers works by using standard interfaces. There is no direct communication between non-adjacent layers. Also one layer does not interoperate with other layers. The basic example of layered protocol is the ISO/OSI model which has a seven-layer protocol stack. The ISO/OSI model has a modular structure. It models and classifies the different network functionalities and services in each layer. The layered architecture has several advantages: first, the layered architecture is flexible. By using the standard interface, the replacement of one old layer does not affect other layers, which means there is no need to modify the rest of the network stack when one layer is updated. Second, because layers are separated, one layer does not affect each other's performance.

However, layered protocol design is not efficient in wireless mesh networks. Due to the properties of dynamic nature, multi-hop communication and no central instance, wireless mesh networks are significantly different from traditional wired networks. In wireless mesh networks, the performance of one layer depends on the other layers. For example, in the physic layer, the physical rate also affects the capacity of the link. The capacity of the link reduces when the physical rate is dropped [2]. In wireless mesh networks, each wireless transmission can be heard by the neighbors and is sensed by them as for their ongoing packet transmissions. That means the performance of the links in wireless mesh networks is not independent of each other. The layered protocol design ignores the interactions between

layers in wireless mesh networks and causes poor network performance. For example, in wireless mesh network, the higher the packet transmission power can cause the higher the transmission rate. At the same time, the increasing the transmission power increases the interference to other nodes too. Therefore the power control is tightly coupled with both the physical and medium access layers. Another example is TCP protocol does not work efficiently in wireless networks due to the properties of multiples hops and dynamic nature [3]. For example, route failures happen frequently due to dynamic nature of the wireless channel and mobility of the nodes. This will result in different round trip times (RTTs), which can affect the congestion control in the transport layer. TCP may assume that the packet drops are caused by congestion and then starts congestion control, thus reducing throughput of the flow.

These examples show that sharing and exchanging information with protocols is useful to gain better knowledge of each layer. Furthermore, layers should work together to improve the performance in wireless mesh networks. One example is the local congestion control mechanism suggested in IEEE802.11s draft, which enables MAC and route protocols work together to support congestion monitoring and signaling. Another example is the HWMP for wireless mesh networks, which has an interaction between routing and MAC/PHY layer to find a better routing path. However, the ideas of collaboration between the different protocols are not supported by layered architecture. The layered architecture can not support new modes of wireless communication in protocol design. Therefore it is necessary to introduce the cross-layer design for wireless mesh networks.

2.2.2 Basic Architectures for Cross-layer Design

Comparing to layered protocol, cross-layer design refers to the protocol design where interactions exist between different layers. With cross-layer design the layers share information with each other and collaborate with each other to improve network performance accordingly. As shown in Figure 2-8, [11] suggests three basic architectures for cross-layer design: direct communication module, a share database module and completely new abstractions module. Direct communication is the way that each layer communicates with other layers directly and allows runtime information sharing between layers. With this proposal, the variables at one layer are visible to the other layers at runtime. A share database approach proposes a common database that can be accessed by all layers. This common database provides the service of storage/retrieval of information to all the layers. The completely new abstractions implement a new way to develop the protocols. This approach

allows rich interactions between the building blocks of the protocols and there is no more layers in the protocol stack.

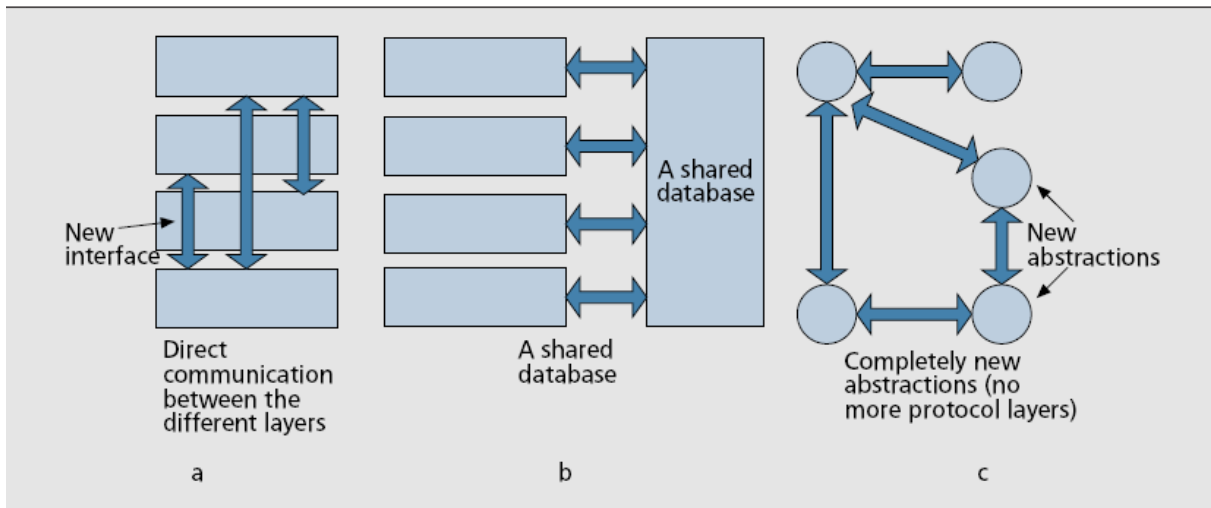


Figure 2-8 Proposals of Cross-layer Architecture [11]

Understanding and exploiting the interactions between different layers of the protocol stack is the core of the cross-layer design concept. With cross-layer design, all layers can share information through new interfaces. Cross-layer design also allows layers to exchange state information in order to improve performance. Variables of layers are accessible to all communication layers. This character enables each layer to have a global view of the constraints and characteristics of the network. Therefore the performance of one layer influences the other non-adjacent layer. Moreover, the network protocols are jointly designed and integrated in cross-layer design. [10], [17] present cross-layer designs to optimize the network performance in wireless networks. These examples show that cross-layer design can allocate the resources efficiently and improve the network performance.

2.2.3 Cross-layer Monitoring

Cross-layer monitoring means observing and collecting the value of metrics of each layer during the transmissions, and providing a function that enables information sharing of layers in the protocol stack or the layers of neighbor's nodes. Cross-layer monitoring is based on cross-layer protocol design. It uses cross-layer architecture. The main goal of cross-layer monitoring is information sharing. Comparing with cross-layer protocol design, cross-layer monitoring does not need a novel protocol stack. It only extends the existing protocols to provide data collection functions and interfaces for information sharing and distribution. The benefit of cross-layer monitoring is twofold: first, the information sharing means that only

minor modifications to existing models are needed. Second, it provides rich interaction information for layers which can be used for network performance optimization.

The cross-layer monitoring mechanisms can be classified to time-driven cross-layer monitoring and event-base driven cross-layer monitoring mechanisms. Time-driven cross-layer monitoring mechanism observes and updates information within a regular time interval regardless of changes to network utilization or network environment. Event-base driven cross-layer monitoring mechanism updates information for layers or nodes by using thresholds. The thresholds are based on monitoring requirements. If the current value of information and the previously value are within the same threshold, update does not occurs. If the values fall into different intervals the previously value of information is considered out of date and an update occurs.

In this thesis, we implement a cross-layer monitoring module based on a cross-layer framework. The reason of using a cross-layer framework is twofold: first, we focus on cross-layer monitoring mechanism implementation. And these mechanisms can be done within a framework. Therefore we do not need to design a novel cross-layer protocol. Second, by choosing a widely used cross-layer framework, the cross-layer monitoring module can be extended quickly for future work.

2.3 Related Work

Several works have studied on cross-layer design in the past. [3] discusses the various cross-layer feedback possibilities and benefits on mobile wireless devices. To improve application performance, it is essential that every layer tunes its parameters accordingly based on the cross-layer feedback. In [3], they also present a survey and introduce three main architectures for efficient cross-layer feedback. [11] suggests a definition for cross-layer design and discussed the basic types of cross-layer design with examples drawn from the literature.

[12] proposes JANUS, a framework for distributed monitoring of WMNs. JANUS is based on the SNMP management framework but the architecture is distributed. Furthermore, the implementation of the module combines a structured peer-to-peer overlay network (Pastry) and a scalable group communication system (Scribe). Every node consists of four components: Mesh node, JANUS Agent, Mesh knowledge base (MKB) and JANUS client. The basic process is described as follows: first the JANUS client sends a query to the Agent. The Agent replies with the MKB object tree accordingly. The client updates the MKB and then publishes them on the Scribe ring. JANUS collects network information at the link layer

and the network layer. And it shares the information to all nodes in the network. However, the prototype uses Pastry as a reference platform and JANUS is implemented based on Scribe. This architecture makes it difficult to extend to common cross-layer monitoring aspects.

In [13], a light-weight monitoring module is developed to measure current network/traffic conditions and to estimate end-to-end path delay. A centralized controller uses a measurement-based admission control (MBAC) scheme to keep track of monitored metric and calculate end-to-end path delay. The MBAC uses a cross-layer monitoring for the module. Each node tracks the per-hop delay during the transmission and sends it periodically to the network layer central controller. The central controller maintains a database of the per-hop delays. At the same time, to discover and maintain the topology of the network, [13] uses a “Hello” message to propagate any changes in their routing table. With the measured per-hop delays and the discovered topology the central controller estimates the end-to-end delay for a given path and makes the admission control decisions accordingly.

[14] presents Xian, a Cross-layer Interface for wireless Ad hoc Networks. The design enables interactions between the IEEE802.11 MAC layer and upper layers. That means MAC layer can communicate with its upper layers directly. The communication is based on a request/response model. XIAN implements one function for each selected metric. The architecture of Xian consists of three components: The Kernel Space Xian Interface (KSI), the User Space Xian Interface (USI), the Xian Information Transport Module (ITM) User Space and the Xian Extended Interface (USEI). With these four components, each upper layer above MAC layer gets the metrics directly. Furthermore, the USEI performs measurement function, operation function and comparison function based on measured metrics. Xian can be used as a service by other network layers or system components to access information about configuration and performance of MAC/PHY layers.

[15] proposes SWAN, a stateless network model which uses distributed feedback-base control algorithms to deliver service differentiation in mobile wireless ad hoc networks. SWAN implements a number of the mechanisms to provide rate regulation of best effort traffic. These mechanisms work together between IP and MAC layer. Each node regulates best effort traffic independently by using AIMD rate control algorithm based on the feedback from MAC. By using a probing request to estimate the end-to-end bandwidth, the source node in SWAN performs the source-based admission control for real-time traffic. The admission control rate is established by comparing the end-to-end bandwidth and the required bandwidth. To regulate real-time traffic dynamically, each node in SWAN uses ECN bits to mark a violation during the transmission. When the destination node monitors the ECN bits, it

sends a regulate message to the source. The source responds to the regulate message by reestablishing its real-time session on need. [15] also analyses different metrics, such as MAC delay and busy probability for SWAN.

To reduce the monitoring overheads in wireless networks, [16] evaluates the impact of monitoring overheads during data transmission. [16] shows that even small amounts of overheads can cause large performance degradation in wireless networks. Several different techniques are suggested for reducing monitoring overheads while maintaining the management that needs to be achieved. For QoS provisioning with statistical monitoring, they use a specific delay-base routing algorithm to reduce the number of monitor agents while be able to monitor every link in the wireless network. For reacting monitoring, they evaluate a threshold-base scheme. By using this scheme, sending data is triggered by a threshold. Only when the measured parameter crosses the threshold, the monitoring agent sends out data. Comparing with the periodic sending data, thresholds-base scheme can reduce the monitoring traffic. [16] also evaluates the impact of frequency of reporting monitoring data on end-user's performance. However, the evaluation of efficient monitoring in [16] is based on measurement-base monitoring. Although they use a threshold-base scheme to reduce data traffic, every node has to report measured data to the center sever. The centralized control scheme for resource and fault management is difficult to be implemented in wireless mesh network and it does not work efficiently due to the distributed and dynamic nature of WMNs. [10] presents an adaptive per hop differentiation (APHD) scheme to achieve end-to-end delay assurance in multi-hop wireless networks. The APHD scheme extends the capability of IEEE 802.11e EDCA technique into multi-hop environments by taking end-to-end delay requirement into consideration. The core component of APHD is Node State Monitoring function, which monitors per class delay (PCD) and share the information with network layer. With the cross-layer monitor approach, APHD makes the intermediate node adjust a data packet's priority level to satisfy its end-to-end delay requirement. However, the cross-layer monitoring function in APHD is limited. APHD is implemented in NS2 for a single proposal of providing end-to-end delay assurance in IEEE802.11e multi-hop networks. The architecture is lack of cross-layer architecture design. Only network layer can share information from MAC layer and only one cross-layer metric (PCD) is used in APHD.

[17] proposes a cognitive resource management framework, CRM, which is based on cross-layer design and enables the distribution of information gathering and decision making across the network. CRM is multi-functional software, which has a modular structure. It carries out cross-layer optimization using a Toolbox. The Toolbox consists of different modules that

provide various optimizing techniques (i.e. genetic algorithm or statistical learning). The core design of CRM is based on Unified Link-Layer API (ULLA) [18]. ULLA provides an interface between the applications and the network devices. Furthermore, the structure of CRM is distributed and extensible, methods can be added in a plug-and-play fashion.

[19] presents Multi InterfAce Cross Layer Extension for NS-2 (MIRACLE), which is a set of libraries for NS-2 to enhance the functionalities offered by NS-2. NS-MIRACLE provides an efficient and embedded engine for handling cross-layer messages. It also provides a modular structure of the protocol stack in NS-2 that enables users to create multiple modules on each layer in the stack. For instance, multiple network, link, MAC or physical layers can be specified and used within the same node [19].

All of above approaches give us ideas to develop a novel cross-layer monitoring module (CLMM) for wireless mesh network in NS2 with NSMIRACLE extension. However, the CLMM is different from above modules. First, the CLMM monitors the data traffic and handles all the metrics of the layers locally. Second, the values of metrics can be shared with the layers. Third, both time-driven and event-driven mechanisms are implemented in the CLMM. That means it updates the metrics of each layer periodically or based on the thresholds. Furthermore, the CLMM has a modular structure and it is configurable. The modular structure can be easily extended for other network aspects. The configurable feature allows us to change metrics and events without reprogramming. The CLMM module is implemented in NS2 with NSMIRACLE. Although NSMIRACLE can handle cross-layer messages, NSMIRACLE lacks cross-layer monitoring modules for wireless mesh networks. Our goal is twofold: First, the CLMM provides functions to monitor and distribute metrics for all layers. Second, the CLMM provides a cross-layer monitoring framework for NS2 with NSMIRACLE extension, which can be easily used and extended for wireless mesh networks.

3 Cross-Layer Monitoring Module for Wireless Mesh Networks

3.1 Introduction

In this chapter, we present the design of the Cross-Layer Monitoring Module (CLMM) for wireless mesh networks. The CLMM is the central instance within a mesh node for gathering and distributing the information. The main features of the CLMM are summarized as follows:

- It has both time-driven and event-driven functions.
- It has a flexible structure to define new events and their process functions for the CLMM.
- It has Layer-Interface functions to each layer. Therefore every layer can communicate with others by using general message.
- It can define its own interfaces. Also it can define a new interface by inheriting or deriving other interfaces. Every layer has its specific interfaces to handle the messages of the layers.
- Each layer has the list of its metrics. Each metric encapsulates all the information it needs, and it can be updated automatically within the layer.
- It has a storage module that can store values of the monitored metrics.
- It updates the metrics of each layer periodically or based on the thresholds.
- Not only does it output the last obtained value of the metrics directly, but also it calculates and stores the average values for those metrics.
- It has an action function to compare the values of metrics (i.e. queue length) with the thresholds, and trigger an action accordingly when the values are out of the thresholds.

A structure of module is presented in our design based on the architecture of NS-MIRACLE. Also, we extend NS-MIRACLE code to implement metrics of layers for the monitoring module and enable NS-MIRACLE to collect, process and distribute the cross-layer messages.

3.2 Architecture of the CLMM

The architecture of the CLMM is shown in Figure 3-1. The CLMM has eight basic function modules. These function modules are the Monitor, Layer-Interface, Message Router, Action, Storage, Routing and Trace module. We will introduce the Monitor, Layer-Interface, Message Router, Storage and Trace module function in the following sections. Because of time limits, the Action, Routing and configuration file have not been implemented in this thesis, we will discuss the design of Action, Routing function and the Configuration file in section 6.2.

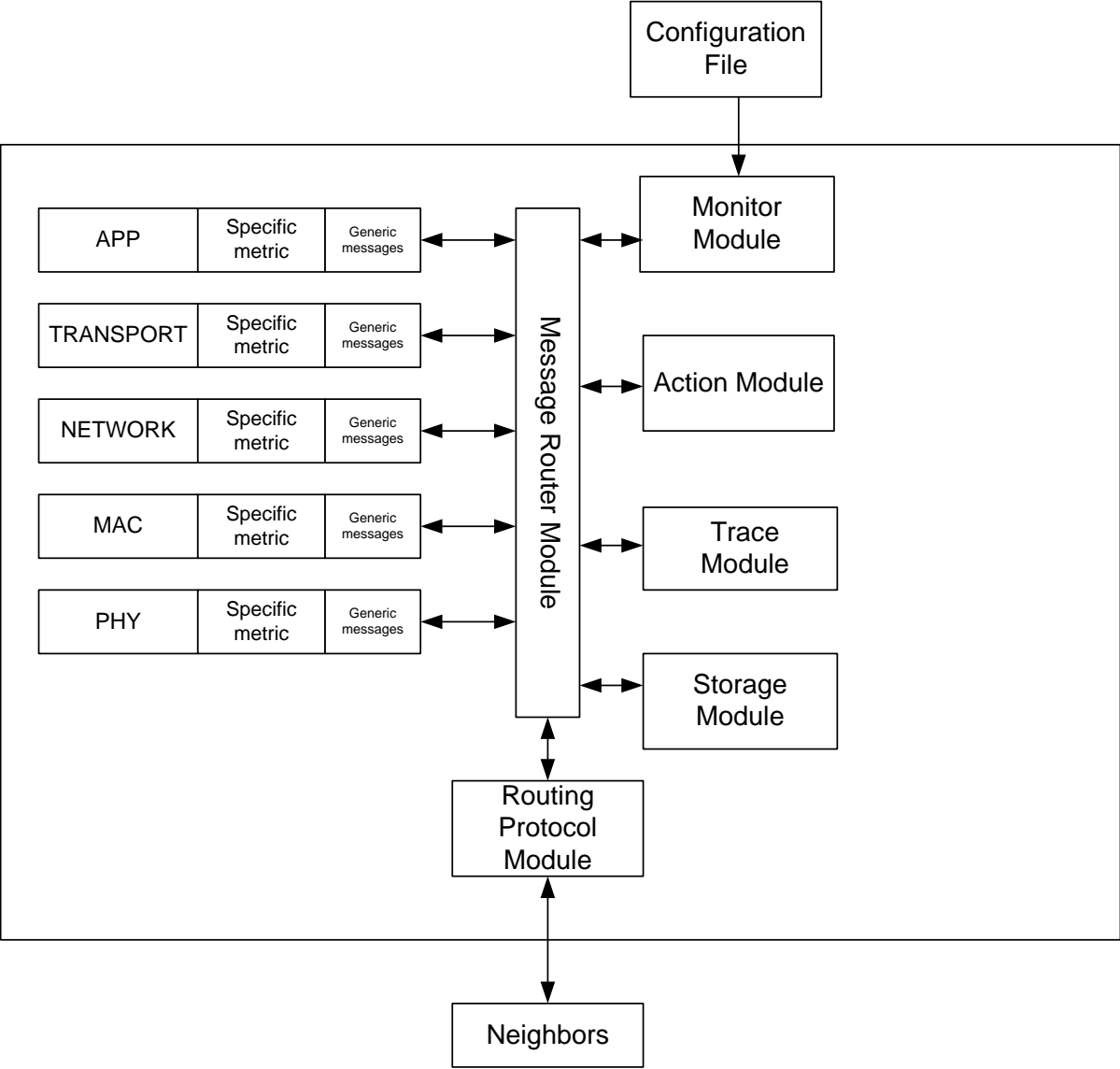


Figure 3-1 The Architecture of the Cross-Layer Monitoring Module (CLMM)

3.2.1 Monitor Module

The monitor module is the core part of the CLMM. It performs the cross-layer monitoring for the CLMM. It implements a time-driven mechanism and event-driven mechanism. Furthermore it has a calculation sub function which calculates the average values for the metrics. After the calculation it stores the results to the storage module. At that time, the values of the metrics are ready to be retrieved or delivered. It also has interfaces which respond to Action Module, Messages Process Module and Layer-Interface Module. The main functions of this module are described as follows:

- It determines when the CLMM starts or stops the cross-layer monitoring. It performs event inheriting, interface inheriting and message inheriting when it initializes the CLMM.
- It performs two monitoring mechanisms for the CLMM: time-driven mechanism and event-driven mechanism.
- It provides interactions between the layers that enable the communication between the layers.
- It provides an error control function for the CLMM. All the errors are sent to and handled by the monitor module.
- The monitor module is configurable and it can be easy configured by a configuration file.

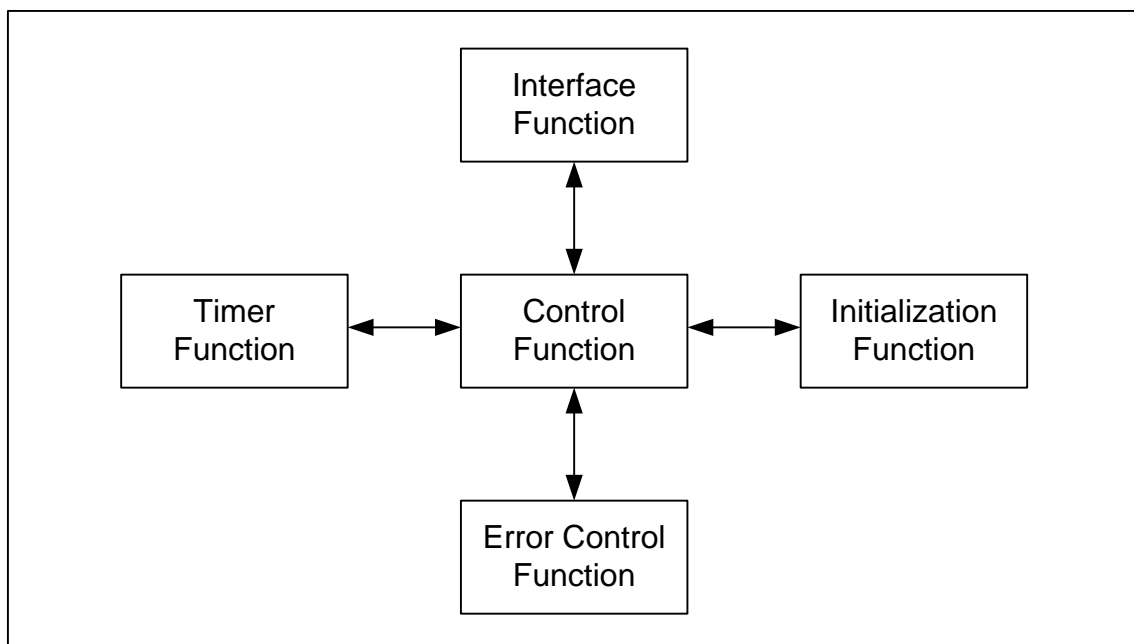


Figure 3-2 The Structure of the Monitor Module

The structure of the monitor module is shown in Figure 3-2. The monitor module has five basic sub-functions, the Control, Interface, Timer, Error control and Initialization sub-function. In following sections we will introduce the Control, Interface, Timer, Error control sub-functions. In section 6.2, we will discuss the design of the Initialization function.

3.2.1.1 The Control Function

The Control sub-function provides the following functions:

- It implements the cross-layer monitoring for the CLMM and it provides interactions between the layers. With cross-layer monitoring, the layers communicate with each other and exchange their metrics as they need. Furthermore, each layer has a function that sends out messages carrying the values of metrics (i.e. the value of sending rate) to the monitor module.
- It performs two monitoring mechanisms for the CLMM: time-driven mechanism and event-driven mechanism.
- It starts the message query for the CLMM. For every message query, the process is done by the following (shown in Figure 3-3):
 - At the beginning, the Monitor module broadcasts a monitor message to the message router module. The message router transfers the query message to the Layer-interface module.
 - When the layer receives the message, it transfers the message to the interface to handle the message.
 - When the message has been processed, the layer sends an interface message to the control function. Then the monitor module identifies each interface and starts to send a query message to the layers.
 - When the layer-interface receives the query message, it starts to retrieve its metrics directly. Then it sends the query results back to the monitor module.
 - After the CLMM has started, the monitor module can also send a query message to the storage module directly. The sequence diagram is shown in Figure 3-4. The storage module sends the specific metric messages back to the monitor module accordingly.
- It processes the routing messages which come from other nodes and responds to them.

- It stops monitoring when the Error Control function reports errors and outputs the errors for debugging.
- It also has a function that processes requests of other modules (such as the action module) and responds to them accordingly.

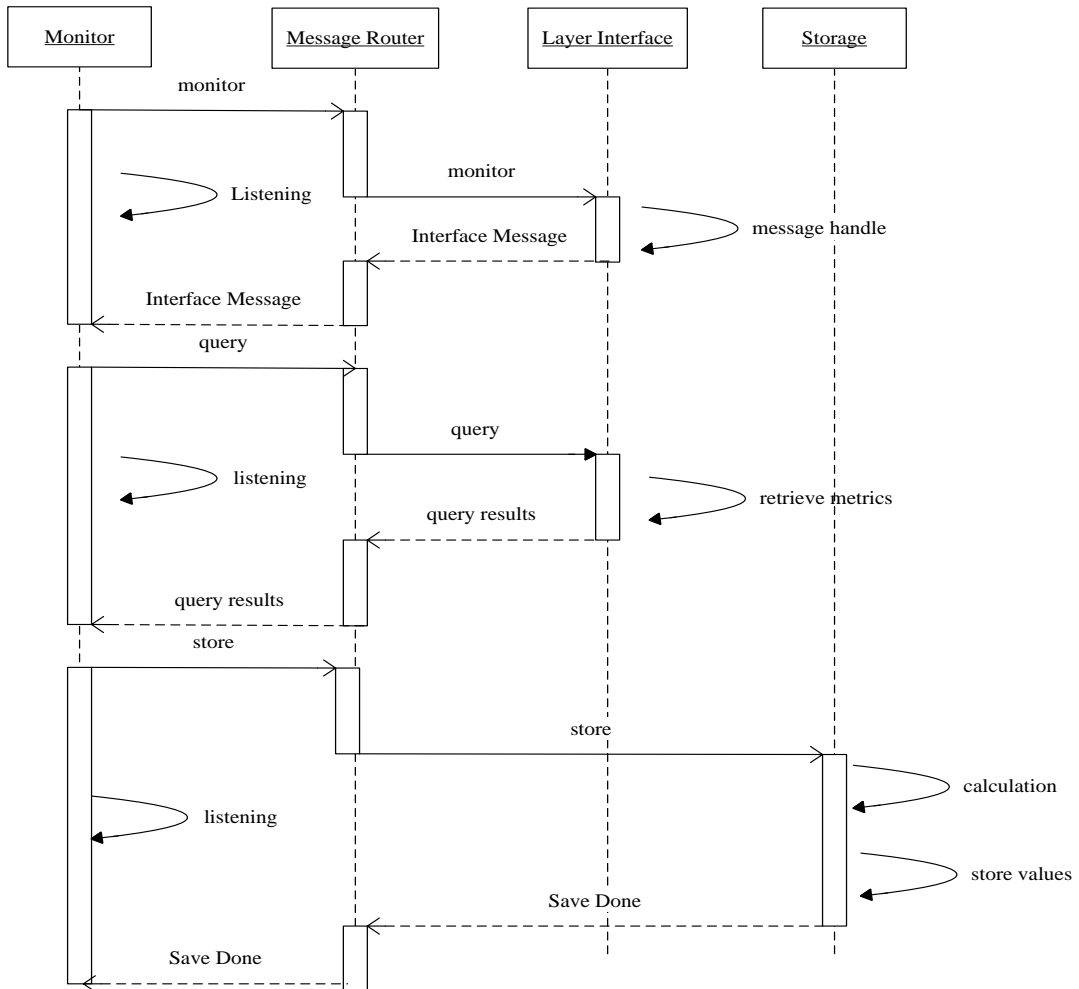


Figure 3-3 Sequence Diagram of Query Message to Layer-interface Module

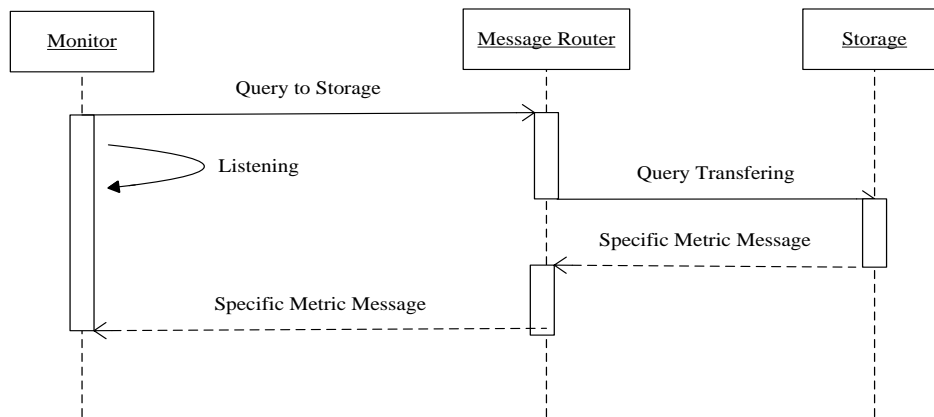


Figure 3-4 Sequence Diagram of Query Message to Storage Module

3.2.1.2 The Interface Function

The Interface sub-function provides interfaces to the Action function, Message Process function, Layer-interface function and the configuration file that enables communications between these functions.

3.2.1.3 The Timer Function

The Timer provides timing function for the Control Unit. When all the functions are enabled and ready, it carries out the monitoring action by period.

3.2.1.4 The Error Function

The Error sub-function handles all of the errors for the CLMM. It outputs the errors for analyzing and debugging. With the Error function, the CLMM monitors the status of the monitoring, halts and quits the monitoring according to the error messages.

3.2.2 Messages for the CLMM

3.2.2.1 Message Types

With messages, the CLMM not only carries out the cross-layer monitoring, but also gets the results (the values of the metrics) it monitored. In the CLMM, messages for the CLMM are categorized into three types: the MONITORMSG, the INTERFACEMSG and the ROUTINGMSG types. To identify these messages, each kind of message is assigned to a unique identity number.

- As shown in Table3-3, The MONITORMSG comes from the monitor module. These generic process messages carrying operating commands are used for message operations during the simulation, such as message query, response and delivery.
- The INTERFACEMSG messages come from layers. The specific metric messages are specified by the sub types and used to identify the metrics of the layers. Each layer has its own metrics. Metrics of different layers are different. Therefore every layer has its own metric messages. And each metric message responds to one unique metric.
- The ROUTINGMSG type identifies the messages of the routing module. The routing message is used to delivery the information of the source to its neighbors. The information is the values of metrics, specific signals (such as a congestion signal) or generic process message. With the information the node’s neighbors know not only the status of the source node, but also the status of the link between them. And it reacts by adjusting the parameters according to the messages.

In CLMM, every message has its message type and extended message type. With the extended message type, a new message can be defined. For example, MONITORMSG is a monitor message. However, it can also be defined as an internal message for several proposals. Furthermore, a message can be used by different modules. In CLMM, we can also define a new message by inheriting or by extending the existed message types. For example, if we want to define a TCP message, we can derive it from the super message class. All the other modules will recognize this new message. Therefore the extended type makes it easy to define its new messages.

Number	name	type
1	Generic process message	MONITORMSG
2	Specific metric message	INTERFACEMSG
3	Routing message	ROUTINGMSG

Table 3-1 The message types

3.2.2.2 The Structure of the Message

The structure of the messages is shown in Figure 3-5. The first is the message type, which identifies what kind of message they are. The next parameter is the sub-type, as shown in Table 3-2, there are two kinds of sub-types, Mtype and event_type. Each sub-type has a set of messages. The third parameter is dtype which defines the type of the destination (UNICAST or BROADCAST). The fourth is the source id. To enable communication between the layers

or functions, each message sent by the source to its destination should have its source ID and destination ID. The source ID and destination ID indicate where the messages come from or go to. If the message is a UNICAST message, the dtype carries the destination ID. If the message is a BROADCAST message, the dtype carries the layer id. The source ID and destination ID are allocated by NSMIRACLE CLMessage class and are traced by CLMessageTrace class. The last parameter is used to carry the values of the message.

type	subtype	dtype	source	value
------	---------	-------	--------	-------

Figure 3-5 The Structure of Message for the CLMM

Mtype	event_type
toall	queue
tomonitor	macinterface
answer	macpktsend
query	macpktrcv
queryresult	macbusytime
update	macbusyperiod
onevent	phy
error	ll
	routing
	forall

Table 3-2 The Sub-types of the Messages for the CLMM

3.2.3 Layer-Interface Module

The Layer-Interface module provides interfaces for data communication between the layers and the Monitor module. As shown in Figure 3-6, every layer has its Layer-interface. Each interface of the layer has its specific metric message and generic message process sub-function. The generic process message function processes query message and delivery message, and responds to these requests accordingly. The specific metric message function processes the metric messages that generated by this layer (i.e. MAC queue length). When the CLMM starts monitoring, the monitor module sends query messages to different layers. When a Layer-Interface receives a query message, first it checks if it is a broadcast message. If it is not a broadcast message, then it checks the destination ID to find out if the destination ID matches its location ID. If the ID does not match, then it drops the message. Otherwise it

processes the query messages and responds by sending its specific metric messages carrying the values of metrics to the monitor module.

Providing interfaces for the layers has two fold:

- First, using interfaces can minimize dependencies between the layers and the module. Interfaces can hide properties and make it easy to be understood and implemented.
- Second, independent interfaces make it easy to be modified for different goals in future work.

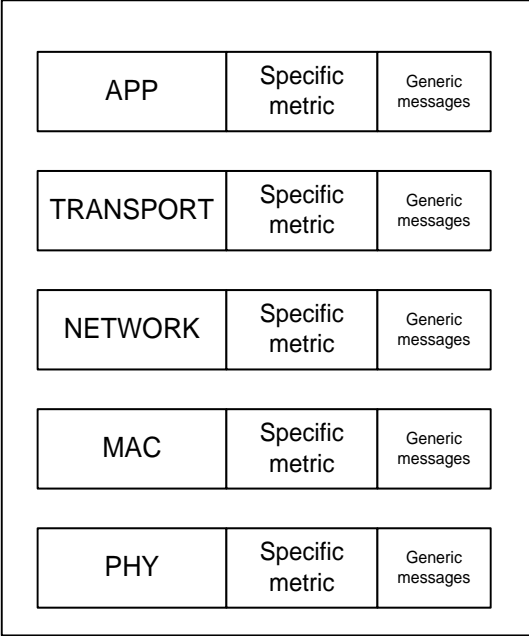


Figure 3-6 The Structure of the Layer-Interface

3.2.3.1 Event-Driven Mechanism for the CLMM

The layer-interface module has a function that every layer can define its event and its event handler. Every Layer-interface defines its own event and has its own event handler to handle its event. Every time when a new event is defined or derived from an event, a new event handler function is added for the new event. For example, if a queue event is defined in MAC layer, a queue event handler is added in MAC layer interface to handle this event.

To implement the event-driven mechanism in the CLMM, we define events and event-handler functions in layer-interface module. We set up the thresholds of some specific metrics (i.e. queue length or busy time) in the event handler functions. The layer-interface updates the values of the metrics and compares the values of the metrics with the thresholds based on the events. If the value is between the thresholds, the event handler function does nothing but keeps on updating and comparing. If the value is upper or lower than the thresholds, the event handler function triggers an action according the configured event-driven mechanism. For

instance, if the value of queue length reaches the threshold, the action module delivers a message to the layers.

The event handler function supports event signaling function. The event signaling function sends out a signaling message to the layers or neighbors when a threshold is exceeded. For example, when the MAC queue length reaches its maximum value that the MAC layer cannot handle, a queue overflow event happens, the event handler function sends out a message to the layers. Therefore other layers know the state of MAC queue immediately.

3.2.4 Message Router

The message router function works like a router. But instead of delivering packets, it delivers all different messages for other functions from sources to their destinations. It supports broadcast and unicast types. It also transfers the routing messages to IP routing protocol. In fact, the message router provides a data channel for the CLMM. All the messages are transferred through the message router. As described in section 3.2.3, all the messages are categorized and each message has its ID. Therefore it is easy for the message router to deliver messages to the destinations.

3.2.5 Storage Function

The Storage function provides a data pool for the CLMM. It has a small size database. This database stores all the values of the metrics sent by the monitoring module during the simulation.

The storage function makes it easy for other functions to retrieve the values of the metrics. And it also reduces the retrieving time. That means the action function can take less time to get the data and it takes an action more quickly. Time is a very critical sector to the action function. The less time it spends the more accuracy it gets.

When the storage function receives a message, it checks categorized types. If the message is a specific metric message, it stores the value of the metric to its database. When the data has been saved, it sends a “save_done” message to the source. If the message is a generic process message, such as a query message coming from the action module, it retrieves the value of the metrics accordingly and sends the result back to the action module. For example, to monitor whether the values are lower or above the thresholds or not, the action module retrieves the values of some metrics (i.e. busy time) from the storage module periodically. First the action module sends a query message to the storage module. Receiving the request, the storage module retrieves the values from the database and sends the results back to the action module directly. The Data types for the database are shown in Table 3-3.

Number	Types
1	Metric type
2	Event type
3	Metric name
4	Original value
5	Timestamp_Original
6	Average Value
7	Timestamp_Aver

Table 3-3 The data types for the database

3.2.6 Trace Module

The trace module provides the trace files that record all the information generated by the CLMM during the simulation. With the trace module, we can not only analyze the results of the metrics that CLMM monitored, but also get the message information of the events.

3.3 Extensibility of the Framework

The CLMM is implemented in NS2 with NSMIRACLE extension. Based on dynamic and modular characters of the NSMIRCALE, the main character of the CLMM is modular extensibility. This framework provides interfaces for the monitor module. With these interfaces, the CLMM can be used as a prototype of cross-layer monitoring. Based on the prototype, it can also be extended to provide new functionalities. In this section, we will introduce these characters of CLMM briefly. The basic contents of NS2 and NSMIRACLE [19] are introduced in section 4.1 and section 4.2.

3.3.1 Module Extensibility

Based on NSMIRCALE, all modules of CLMM are easy to be extended. NSMIRACLE uses Module instead of NS2 Agent [21]. Thanks to the Module class, modules can communicate with each other (in same, above or bottom layers) and handle or modify packets. All the modules are derived from a base class Plugin which is designed to attach any module direct to the Node-Core external from the stack of modules [19]. We modify the Plugin class by adding new interfaces and functions. By using the virtual functions, we can use these new interfaces and functions in derived classes and functions. Interface extension and function extension can be easily done without modifying the father class. That means new methods and new configuration can be realized easily by using override or overload techniques. In CLMM, all

these configurations are done by the modified PlugIn class. At the same time, the configuration for every module is specific. For example, one module needs a sendup () and senddown () functions, another module has messages and message handle interfaces. By deriving from the PlugIn class, the sub-modules can select and configure their own functions. We also integrate the methods for adding and managing the interfaces. Therefore the CLMM can add its own interfaces and functions in need.

3.3.2 Event Extensibility

In CLMM, an event is defined as an instance event. That means the event must be handled immediately. All the events for CLMM are derived from NS2 base class EVENT [21]. That makes the events are scheduled and handled by NS2 scheduler. In NS2, an event is triggered after duration of a time parameter. If the parameter is zero, then the event is an instance event. The CLMM has a base event class and event handle class. The base event class defines one basic event and one event handle function. With this base event class, new events can be extended by inheriting and deriving from the base event class. The new event handle function can also be added in a same way. The events are added or generated in the modules. When the condition is satisfied, an event occurs and it notices NS2 to schedule the event. NS2 schedules the event and notice the event handle function to process it. For example, based on the base event class, we can define a new event and event handle function for MAC layer by setting up a new event type. If the MAC event occurs and is thrown out, the MAC interface catches it and handles the event based on the event type. The character of event extension makes the CLMM flexible and easy to define a new event.

3.3.3 Message Extensibility

In NSMIRACLE, all messages are different. Every message has one global CLMessage type. New messages are added by deriving from the CLMessage class. The CLMessage types are initialized and registered automatically when the NSMIRACLE is loaded. In CLMM, the messages are different. Although the messages of the CLMM are based on NSMIRACLE CLMessage class, the structure of messages for the CLMM is simplified. Compared to the messages of NSMIRACLE, the messages implemented for the CLMM are categorized to three types (as described in section 3.2.2). The CLMM has a message type list and message sub-type list. The sub-type list includes all message sub-types. These sub-types define which metrics of the layers the messages belong to. By using the message type and the sub-type, the message processing of the CLMM is simpler than the NSMIRACLE CLMessage. The less message types means using less resource. Furthermore, new messages can be added and

extended easily by deriving from the base CLMessage class. To add new messages, first we add new message types and sub-types to their type lists. Then, we add message handle functions for these new messages. All the new messages are initialized and registered automatically. To add an INTERFACEMSG message is simpler. For example, if we want to add a message (i.e. network_ett) for network layer, we add a sub-type “network_ett” to the sub-type list and a message handle function for this message. By this way, message extension for the CLMM becomes very flexible and efficient.

3.3.4 Metrics Extensibility

In CLMM, each metric is encapsulated as a container. The container encapsulates data and the operation methods. The calculation methods and algorithms are different due to the different metrics. And they are implemented within the container. Thus the data and the operations methods are bond together to realize the data hiding. The only interfaces of metrics that connect to outside are setvalue(), getvalue() functions and the parameter list. Via these interfaces, the metrics work like “black boxes”. To get the results, what we need to do is to input the data. And then they can be calculated independently (without being affected of other modules). The advantages of using “black box” are summarized as follows:

- Every black box has its parameter list and operation methods. The communication between the metrics and modules are via interfaces. To get the results, only data need to be input. From outside view, all the “black boxes” are the same. All metrics are calculated by their parameter lists and connected by their interfaces. When the metrics are stored, the storage module does not know (and does not care about) what it has stored. By this way, metrics can be highly independent.
- Since the calculation methods and algorithms are encapsulated in the “black box”, the operation methods for the metrics are independent and specific too. And they are easily to be changed without changing other modules. For example, if we want to calculate a metric based on different models, all we need is to modify the operation methods and the parameter list without changing other parts of the modules.
- Using “black box”, the metrics are also flexible to be extended. The metrics can be inherited and derived. New metrics can be created quickly by inheriting and overloading their operation methods.

3.3.5 Layer-interface Extensibility

In CLMM, layer-interfaces are specified for the layers. In fact, each layer-interface is a thin client of the monitor module which is hooked to the layer. The layer-interface is a friend class of the Layer. Therefore it can access to the layers and fetch the information of the layers. It takes charge with the communication between the layers and the monitor module. It also handles the messages and events of the layers. The advantages of using layer-interface are twofold:

- The modification for the communication between each layer and the monitor module is reduced to minimum.
- With the layer-interfaces, each layer can implement its own metric handle functions and event handle functions. Handling metrics and events within the layers means it uses less resource and takes less time to react.

All the layer-interfaces are inheriting from the base Interface class. The base Interface class has a basic message handle function and event handle function. The new layer-interfaces can be added by deriving from the base Interface class. For example, in NSMIRACLE, two kinds of MACs are implemented: the dei802.11mr and NSMIRACLE 802.11 MAC. If we want to use these two MAC layers for the CLMM, we simply create two layer-interfaces for them and add the layer-interfaces in the simulation scripts. By this way cross-layer messages can be implemented for the layers without modification of the layers.

3.4 Metrics

3.4.1 Introduction

As described in section 2.5, different sets of metrics are used for different research proposals. The cross-layer monitoring design for wireless mesh networks should integrate the inter-layers and inter-nodes metrics to improve the network performance efficiently. However, [16] shows that too much metrics cause performance decreased. Furthermore, the limitation of software developing environment should be considered too. Because the implementation of the CLMM will be done in NS2 with NS-MIRACLE extension, parameters for the monitoring module should be selected carefully and following two principles:

- First, these metrics can be used to improve the network performance efficiently.
- Second, metrics should be implemented easily in NS2 with NS-MIRACLE.

To select metrics for the CLMM, we first examined all metrics used in the proposals presented in section 2.5. Then we made a list of these metrics and selected metrics for the monitoring modules according to the above two principles. The metrics we have selected are listed in Table3-4, Table3-5 and Table3-6. These metrics will be implemented for the monitoring module. However, not all metrics will be implemented in the monitoring module at the same time. Not only for time limit reason, but also for the reason of software overhead that should be considered during the software design section . Not all metrics are needed for one specific research proposal. Therefore metrics for monitoring module should be configurable. The configurable feature of the module not only makes it easy to implement metrics, but also flexible to add or change metrics if needed.

First, we made a prioritization of the metrics implementation. These metrics will be implemented in three phases. In first phase we plan to implement the metrics listed in table1, which are useful and easy to be implemented for the module. These metrics should be finished for this project. The metrics in table2 are more difficult and will be implemented if time permits. The metrics listed in table3 are not supported in NS2 with NS-MIRACLE extension currently, but can be implemented in future work.

Prioritization	Metrics	Layer	Used for	Classification
Phase1	the MAC queue length	MAC	rate control	Frame Rates/Counters
	the average packet send rate	Network	rate control	Per neighbor/link metrics
	the average packet arrive rate	Network	rate control	Per neighbor/link metrics
	the channel busy time	MAC	calculate channel utilization percentage	Frame Rates/Counters
	the number of busy period	MAC	channel utilization	Frame Rates/Counters
	the number of one hop neighbors	Network	link quality	Frame Rates/Counters Metrics

Table 3-4 Metrics that are Implemented for CLMM in Phase1

3.4.2 Classification of Metrics

The metrics shown in table3-4-3-6 are clustered in three groups: Per neighbor/link metrics, Frame Rates/Counters Metrics and Delay Metrics. In this section we describe the definitions of the classifications, and then we describe these metrics in detail in section 3.6.3.

Prioritization	Metrics	Layer	Used for	Classification
Phase2	Chanel idle rate	PHY/MAC	idle channel;	Frame Rates/Counters Metrics
	Channel transmission rate(air time)	PHY/MAC	channel occupied by successful transmission of the node;	Frame Rates/Counters Metrics
	Channel collision rate	PHY/MAC	channel occupied by a collision of nodes	Frame Rates/Counters Metrics
	Channel busy rate	PHY/MAC	busy channel due to activity of other nodes, detected by means of either physical or virtual carrier sensing (i.e.,NAV)	Frame Rates/Counters Metrics
	the number of received data frames with/without retransmission	MAC	aggregated statistics	Per neighbor/link metrics
	the number of sent data frames with/without retransmission	MAC	aggregated statistics	Per neighbor/link metrics
	the number of retransmissions	Link	overhead	Frame Rates/Counters Metrics
	the average channel idle time	MAC	calculate channel utilization percentage	Frame Rates/Counters
	per hop delays	MAC	End-to-end path delay	Delay Metrics
	Average frame arrival rate	MAC	Rate control	Frame Rates/Counters
	the relative signal strength (RSSI) of the last ACK on transmission(per link)	PHY	CCA,link quality measurement(a measure of the RF energy received by the received signal strength level)	Per neighbor/link metrics
	used channels	PHY	Configuration states	Frame Rates/Counters Metrics
	the number of failed receptions (due to queue overrun, bad CRC, PHY errors or decryption problems)	MAC	link quality	Frame Rates/Counters Metrics
	Expected Transmission Count /ETX	Network	link quality/packet loss ratio	Frame Rates/Counters Metrics
	ETT	Network	link quality	Per neighbor/link metrics
	per class delay	MAC	current hop delay	Delay Metrics
	Average frame length	Link/MAC	traffic load	Frame Rates/Counters Metrics

Table 3-5 Metrics that to be Implemented for CLMM in Phase2

Prioritization	Metrics	Layer	Used for	Classification
Phase 3	the number of received frames dropped or with wrong BSSID	MAC	Aggregated metrics	Frame Rates/Counters Metrics
	transmit power	PHY	transmission range adaption(MAC)/power saving	Per neighbor/link metrics
	end-to-end delay	Network	overhead/routing(delay-based)	Delay Metrics
	RTT(per hop)	TCP	threshold-based monitoring	Per neighbor/link metrics

Table 3-6 Metrics that to be Implemented for CLMM in Phase3

3.3.2.1 Per Neighbor/Link Metrics

Per neighbor/link metrics store per neighbor and per link information related to particular transmission at MAC layer [14]. This kind of metric relates the number of received/transmitted data frames or bytes, the relative signal strength (RSSI) or the number of retransmissions. These metrics can be considered as elementary metrics and can be obtained directly from output of trace files during the simulation in NS2. They can be used for calculation of average value of a given metrics (i.e. the average sending rate).

3.3.2.2 Frame Rates/Counters Metrics

Frame Rate/Counter Metrics provide global status on the usage of the IEEE 802.11 network interface during the transmission. For example, the metrics can be: the number of the frame retransmissions, or the number of one-hop neighbors within a time interval.

3.3.2.3 Delay Metrics

Delay Metrics provide the sum of time needed for a packet from source to destination. Delay metrics can be one-way delay metrics or round-way metrics. One-way delay is the difference between the time when the source sends out the first bit of the packet and the time when the destination receives the last bit of the packet. Round-trip delay is the sum of the time needed for a test packet travel from source A to destination B and from B back to A. The metrics can be one-hop delay, or per class delay.

3.4.3 Metrics for Monitoring Module

3.3.3.1 Number of One-hop Neighbors:

The number of one hop neighbors is the total number of one-hop neighbors of the node in a given time interval. The number of one hop neighbors can be counted from a routing table.

3.3.3.2 Average Packet Arrival Rate:

The average packet arrival rate is the number of packets that arrive at the node within a time interval. The average packet arrival rate is varying due to the link quality. The high packet arrival rate can lead to queue overflow when the node can not handle. And that will result in high loss level on the uplink.

3.3.3.3 Average Packet Sending Rate:

The average packet sending rate is the number of packets that sent at the node within a time interval. The average packet sending rate is varying due to the link quality. The high packet sending rate can lead to queue overflow when a mesh point has a high downlink delay due to the increase in retransmission levels or high error rate. And that will result in high loss level on the downlink.

3.3.3.4 Channel Busy Time

The channel busy time is defined to be the amount of time during which a node using the CS mechanism to sense the channel busy [8]. As described in section 2.1.2.1 and 2.1.3.1, the channel is sensed busy as a result of two different functions: physical carrier sense (P-CS) or virtual carrier sense (V-CS). Since all stations are synchronized in single-hop IEEE802.11 wireless networks, the channel busy time equals the duration of a successful packet exchange or the duration of a collision. However, due to lack of coordination in the multi-hop wireless networks, the channel busy time is much longer than that in single-hop wireless networks. [20] computes the average channel busy time as:

$$T_b(i) = T_{idle}(i) [1 - Q(\varphi)] / Q(\varphi) \quad \text{Equation 3-1}$$

Where $T_{idle}(i)$ is the average duration of an idle period, $Q(\varphi)$ is the probability that none of virtual nodes is active.

3.3.3.5 Number of Busy Period

The number of busy period is the number of times that a channel is sensed busy. We denote the number of busy period as num_busyperiod and the total number of the busy period as sum_busyperiod. The calculation of the number of busy period is shown in Figure 3-7.

As described in section 2.1.3.1, when the node starts to initiate a frame, it sets the NAV to the time for which it expects to use the medium. All nodes monitor the wireless medium. If the wireless medium is busy, the number of busy period is incremented by one and the NAV counter is frozen. If the wireless medium is idle, the V-CS checks if the NAV is zero. If not, the NAV is decremented by one and the node keeps monitoring the wireless medium. Before the NAV is zero, the virtual carrier-sensing function indicates that the medium is busy. When the NAV reaches 0, the virtual carrier-sensing function indicates that the medium is idle. The frame is sent out right after the NAV reaches zero.

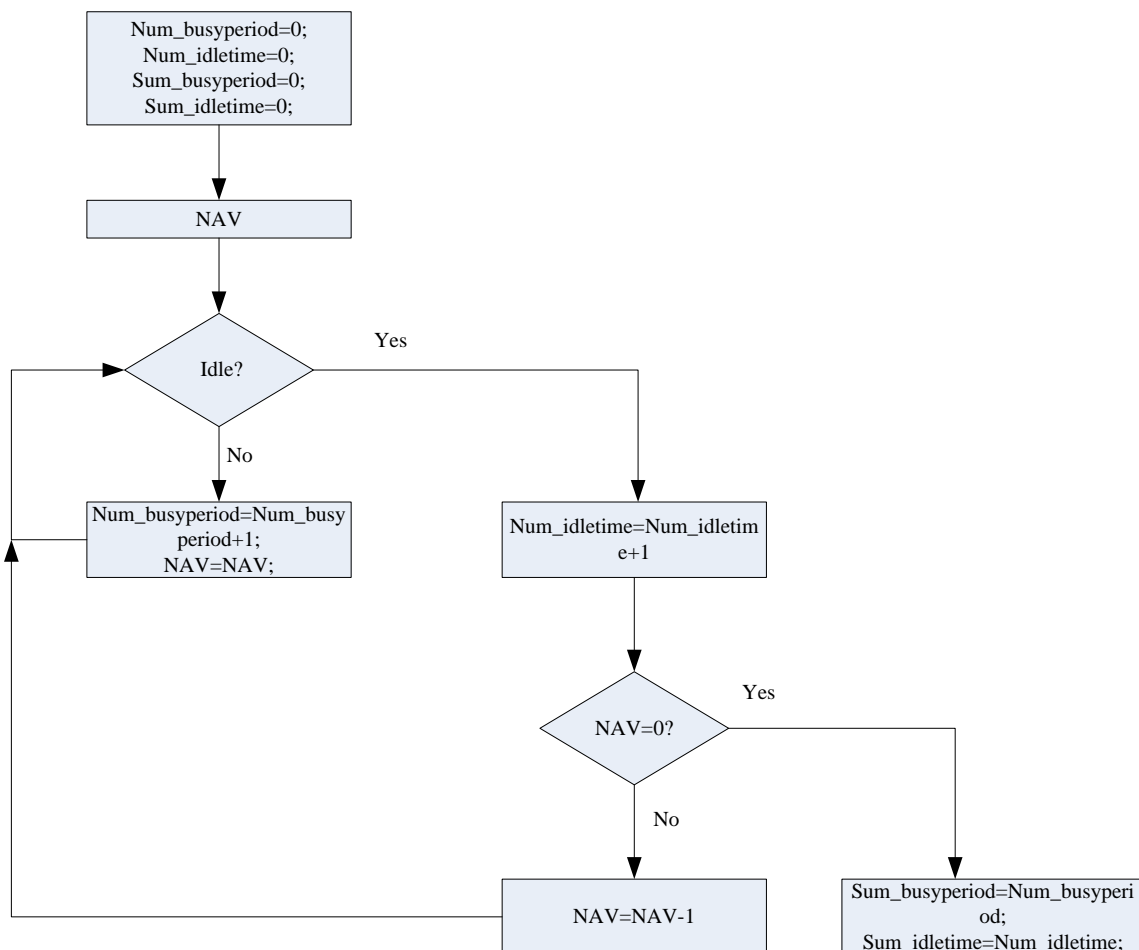


Figure 3-7 THE CALCULATION OF THE CHANNEL BUSY TIME AND THE CHANNEL IDLE TIME

3.3.3.6 MAC Queue Length

The value of the MAC queue length indicates the queue space utilization and how busy the MAC layer is. We use the MAC queue length as an indication of the current network traffic load. When the traffic stream increases, the arriving rate is larger than the sending rate and the queue builds up. If the arriving rate reaches the maximum capacity that MAC can handle, this queue rapidly jams and drops occur. When the traffic is low, the queue length stays at a low level. One local congestion monitoring mechanism is proposed in IEEE802.11s standard draft [5]. Each MP observes the level of congestion based on the queue size. When the traffic increases, the MP can not forward and source data upstream as fast as the incoming rate, congestion occurs. If queue size is sufficient, the MP sends out a congestion signal to its one-hop neighbors.

3.3.3.7 Channel Idle Time

The channel idle time is defined to be a duration time during which the CS mechanism has indicated a channel idle indication [8]. A node senses the channel is idle by using P-CS or V-CS. [20] calculates the average channel idle time as below:

$$T_{idle}(i) = 1 / \sum u_{gu} \quad \text{Equation 3-2}$$

Where the g_u is the activation rate [20].

3.3.3.8 Number of Idle Time

The number of idle time is the number of times that a channel is idle. We denote the number of idle time as $num_idletime$ and the total number of idle time as $Sum_idletime$. As shown in Figure 3-7. The number of idle time is incremented by one when the wireless medium is sensed idle.

3.3.3.9 Channel State Rates:

[20] identify 4 different channel states: idle channel, channel occupied by successful transmission of the station, channel occupied by a collision of the station and busy channel due to activity of other nodes. The occurrence probability of each of the four channel states is specified as follows,

$$\Pi_s = \tau * (1-p) \quad \text{Equation 3-3}$$

$$\Pi_c = \tau * p \quad \text{Equation 3-4}$$

$$\Pi_\sigma = (1-\tau) * (1-b) \quad \text{Equation 3-5}$$

$$P_b = (1 - \tau) * b \quad \text{Equation 3-6}$$

Where P_s is the probability of transmission channel, P_c is the probability of collision channel, P_σ is the probability of idle channel, P_b is the probability of busy channel, τ is the probability that the station sends out a packet after an idle slot, p is the probability that a transmission of the station is not successful, b is the probability that the channel becomes busy after an idle slot due to activity of other nodes [20].

3.3.3.10 Number of Received Data Frames with Retransmissions:

The number of received data frames is the total number of decoded data frames received at the MAC layer within the time interval, including the retransmission data frames. It is a per-link metric, which can be used for aggregated statistics.

3.3.3.11 Number of Received Data Frames without Retransmissions:

The number of received data frames is the total number of decoded data frames received at the MAC layer within the time interval. It does not include the retransmission of data frames.

3.3.3.12 Number of Sent Data Frames with Retransmissions:

The number of sent data frames with retransmission is the total number of data frames that the node has sent within the time interval, including the retransmission data frames.

3.3.3.13 Number of Sent Data Frames without Retransmissions:

The number of sent data frames without retransmissions is the total number of data frames that the node has sent during the transmission, but not includes the retransmission data frames.

3.3.3.14 Number of the Frame Retransmissions:

The number of the frame retransmissions is the total number of the frames retransmitted on a link within a given time interval. The link layer will retransmit the frame a certain number of times until it receives the ACK, before reporting it as lost to the higher layers. The metric indicates the channel condition. [16] shows that the higher retransmission value will result higher end-to-end delay. The number of the frame retransmissions can also be used for automatic rate control mechanism. We denote the number of the frame retransmissions as Num_retransmission and the sum of the frame retransmissions as Sum_retransmission. The calculation of the number of the frame retransmissions is shown in Figure 3-8. When the link

layer sends out a frame to the destination, first it waits an ACK frame from the destination. If it can not receive the ACK frame within an ACKTimeout [8] interval, the number of the retransmission is incremented by one. Before the maximum number of retransmission is exceeded, the sender retransmits until it receives an ACK. When all retransmission attempts have been exhausted or an ACK frame is received, the total number of retransmission is outputted.

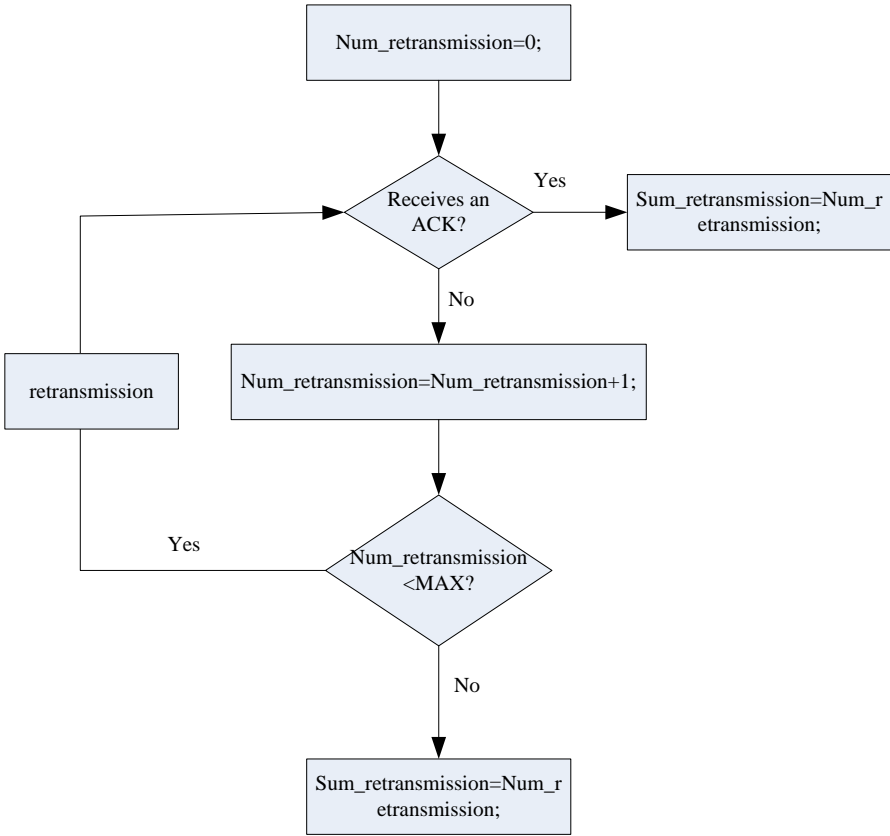


Figure 3-8 THE CALCULATION OF THE NUMBER OF THE FRAME RETRANSMISSIONS

3.3.3.15 Per-hop Delay:

[13] measures per-hop delay for end-to-end delay measurement. In the CLMM, we denote per-hop delay as $D=Ta-Tr$, where Ta is the time at which the MAC layer ACK is received, Tr is the time where the MAC starts the transmission process from the frame to the receiver the first time. The difference between time gives us the total delay involved in forwarding a data frame on that link.

3.3.3.16 Average Arrival Rate of Frame:

The average arrival rate of frame is the number of frames arrived at MAC layer within a time interval.

3.3.3.17 Received Signal Strength Indication (RSSI):

The RSSI is a measurement of RF energy received [8]. It is an optional parameter that has a value of 0 through RSSI Max (0-255, the maximum value is depended on the vendor). RSSI is used for clear channel assessment (CCA) and link quality measurement. RSSI is stored on TX/RX descriptor and is measured by baseband PHY for each individual packet.

3.3.3.18 Expected Transmission Count (ETX):

[24] presents Expected Transmission Count (ETX), ETX is defined as the expected number of MAC layer transmissions that is needed for successfully delivering a packet through a wireless link. Each node estimates the frame loss ratio pf to each of its neighbors within a time interval, and gets an estimate pr of the reverse direction from its neighbor. These values are obtained using broadcast probe packets at the link layer once every second. The pf and pr are the fraction of the probes and the acknowledgments correctly decoded in the last ten seconds. As for a link metric, ETX captures the effects of packet loss ratios. This metric is calculated as:

$$ETX=1/(1-pf) * (1-pr) \quad \text{Equation 3-7}$$

Where the pf is the forward loss ratio and pr is reverse delivery loss ratio.

ETX has the best performance for static networks, but it can not react quickly enough to highly variable and bursty error situations.

3.3.3.19 Expected Transmission Time (ETT):

Because a lower bit rate ends up using the channel for a longer period of time, [25] presents Expected Transmission Time (ETT). The ETT is defined as the expected MAC layer duration for a successful transmission of a packet at the link. The ETT metric captures the impact of link capacity on the performance of the path. It is calculated as:

$$ETT=(s/b)/ETX \quad \text{Equation 3-8}$$

Where s is the packet size, b is the bandwidth of the link.

3.4.4 Exponentially Weighted Moving Average (EWMA)

We use EWMA to calculate the average value for these metrics. For example, the average of queue length is calculated as

$$Q_{av\ n} = w * Q_n + (1-w) Q_{av\ n-1} \quad \text{Equation 3-9}$$

Where w is a weight ($0 < w < 1$), n is the number of observations to be monitored, Q_n is the current queue length, $Q_{av\ n}$ is current average queue length. The delta value of each metric is calculated as $Q_{\Delta} = Q_{av\ n} - Q_{av\ n-1}$.

4 Implementation

We implement the CLMM in NS2 with NS-MIRACLE extension. In this section, we will present the details of implementation. First we introduce NS2 and NS-MIRACLE in section 4.1 and section 4.2. Then we present the classes and functions of the CLMM in section 4.3. In section 4.4, we introduce the process of the implementation.

4.1 NS2

To simulate and evaluate the monitoring module, it is useful to use the open source network simulator, NS-2 [21]. NS-2 is a discrete event simulator written in C++ and Otcl for networking research. It works at packet level and schedules the events such as packet and timer expiration. It provides substantial support for simulation of TCP, routing, and multicast protocols over wired, wireless and satellite networks. NS2 is not a real time simulator. Instead of handling the events at the same time, the centric event scheduler handles events one by one. However it is not a serious problem since the events are often transitory in most network simulations. Nowadays, NS2 is widely used as a standard experiment environment in research community.

4.1.1 NS2 Architecture

The architecture view of NS2 is shown in Figure 4-1. NS2 consists of two kinds of classes: C++ classes and OTcl classes. The C++ classes are used for packet handling and event processing. The event scheduler is the main controller of events. The network components are Node, Link, Queue, etc. Both the event scheduler and network components are written in C++. The OTcl classes provide control and configuration functions. With OTcl scripts we can define network topologies, schedule events or applications that we want to simulate. The tclcl is a language that provides a linkage between C++ and OTcl. The compiled objects are available to the OTcl interpreter through the OTcl linkage. Users design and run simulations by using an OTcl scripts. NS2 takes the OTcl script as an input and produces a trace file as output. The one-one correspondence between classes of these two class hierarchies is shown in Figure 4-2.

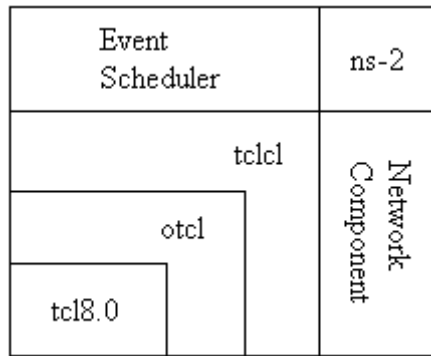


Figure 4-1 Architecture View of NS2 [26]

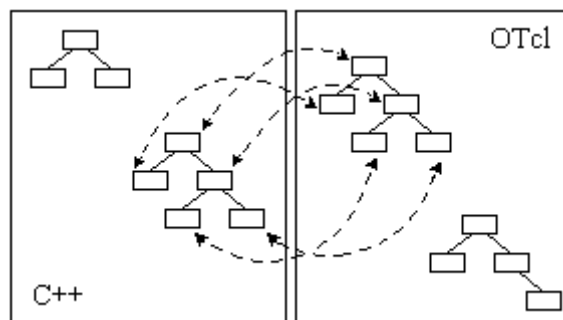


Figure 4-2 C++ AND OTCL: THE DUALITY [26]

Figure 4-3 shows a partial OTcl class hierarchy of NS2. The TclObject is the superclass of all OTcl library objects (scheduler, network components, timers and the other objects including NAM related ones) [23]. NsObject class is the superclass of all basic network component objects that handle packets, which may compose compound network objects such as nodes and links. The basic network components have two kinds of subclasses: Connector and Classifier. A connector performs some function such as receiving and delivering the packet to its neighbor. The function of a classifier is to distribute incoming packet to the correct agent or outgoing link.

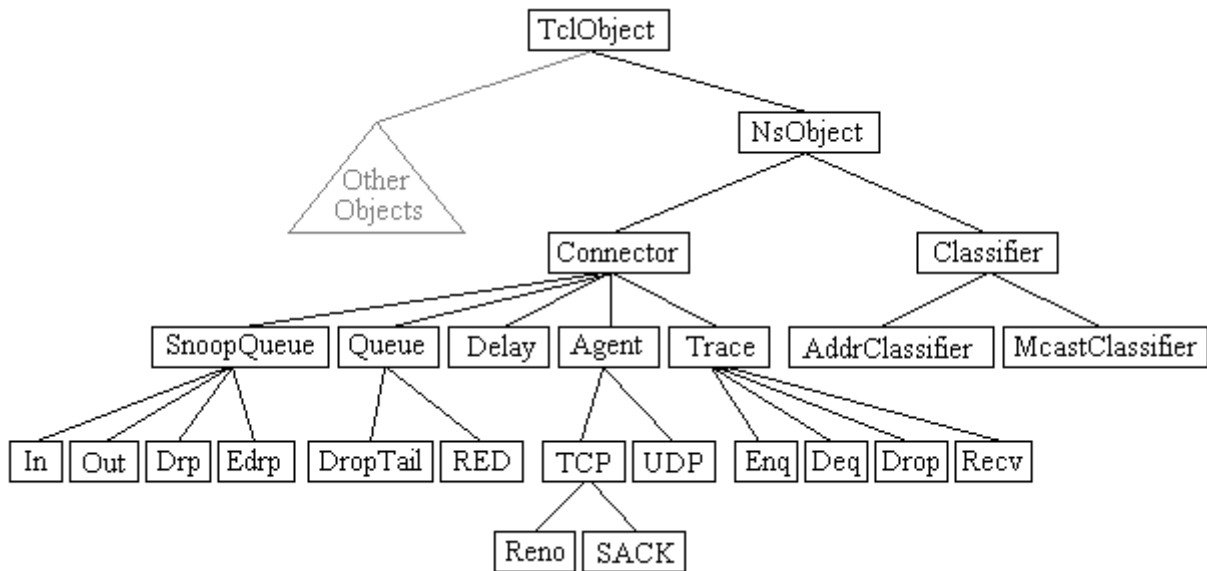


Figure 4-3 Class Hierarchy (Partial) [26]

4.2 NS-MIRACLE

Although multi-rate features are added to the simulator, however, NS-2 lacks flexible tools for handling the cross-layer messages. [19] presents Multi InterfAce Cross Layer Extension for NS-2 (MIRACLE), which is a set of libraries for NS-2 to enhance the functionalities offered by NS-2. NS-MIRACLE provides an efficient and embedded engine for handling cross-layer messages. It also has a modular structure of the protocol stack in NS-2 that enables users to create multiple modules on each layer in the stack. NS-MIRACLE libraries make multi-layer design and interlayer communication in NS2 possible and flexible. For instance, users can set up specified multiple layers in the same node. Cross-layer messages can be exchanged among the Layers. Therefore it is useful to help researcher in implementing any type of cross-layer and multi-technology solutions.

4.2.1 NS-MIRACLE Architecture

NS-MIRACLE has a multi-layer architecture with cross layer communication functionalities. There are a set of new classes that are the basic blocks to model heterogeneous architectures. Figure 4-4 shows the architecture of NS-MIRACLE. The basic classes of NS-MIRACLE are: Module, PlugIn, NodeCore, Cross-layer Message. We will introduce these classes in the following section s.

4.2.1.1 Module

Module is a class designed to contain any protocols or modules placed in stack to follow the OSI model in the designing process of the structure [19]. As shown in Figure 4-4, one or more modules are placed in the same protocol layer. The main function of Module is to communicate with each other (in same, above or bottom layers) and to handle or modify packets. The communication between the modules of adjacent layers is supported by the Server Access Point (SAP). The SAP is a channel for the communication between modules of adjacent layers. SAP is also used for tracing in simulations. All of the modules in the same layer are connected to the NodeCore via Cross-layer SAP (CLSAP). CISAP is defined to deliver the cross layer messages for the modules.

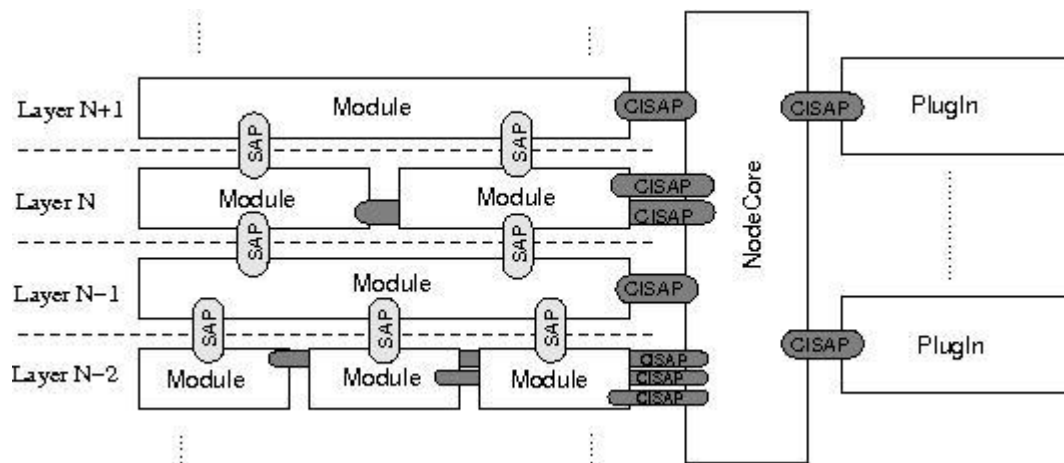


Figure 4-4 An Example of NS-MARICLE Architecture [23]

4.2.1.2 Node-Core and Cross-Layer Message

Node-Core is a class that dispatches all cross layer messages of modules. NodeCore is designed to enable communication among Modules and thus to facilitate cross-layer design. And NodeCore functionality consists of managing information and providing functionalities of common interest for all Modules [23]. With NodeCore the cross-layer messages of the modules can be shared and exchanged in NS-MIRACLE. Also NodeCore maintains the geographical position for each node.

In NS-MIRACLE, the cross-layer message is defined in the CMessage class. There are two message types in NSMIRACLE: the synchronous (UNICAST) and the asynchronous (BROADCAST) CLmessages. The asynchronous CLMessage does not request any direct response answer. The message is directed to all the modules of a specific layer or of the whole architecture. The synchronous CMessages need the recipient answers when the CMessages has been received. The message is directed to a specific module or plugin recipient.

Every message has its source id and destination id. The source id is a unique id that generated during the simulation. If the message is a UNICAST message, the destination id is the id of the destination module or plugin. If the message is a BROADCAST message, the destination id is the layer id or the broadcast address. The CLMessage class has member functions to handle the messages. With the CLMessage class it is possible to define new messages in order to allow cross any type of cross layer communication [23].

These cross-layer messages are exchanged via the CLSAP. The CLMessages delivery is done by the CLTracer class. The CLTracer class has CLMessageTracer classes. When the CLTracer handles the messages, it checks the incoming CLMessage ID and gives it to the correctly CIMessageTracer.

As shown in Figure 4-4, all Modules and PlugIns connect to Node-Core via CLSAP. Therefore all cross-layer messages are exchanged through the CLSAP and are managed by Node-Core. With this structure, cross-layer messages of each layer can be designed and obtained as we need. Furthermore, these messages can be shared and used by different modules. That means the management of the state of common resources or information about the current module status is available in NS-MIRACLE.

4.2.1.3 PlugIn

PlugIn class is the parent class of Module. It is designed to attach any module direct to the Node-Core external from the stack of modules [19]. PlugIn has only the cross layers communication functionalities of a Module. Therefore it is equipped with the CISAP to connect to the Node-Core. PlugIn makes it possible to contain all features that are difficult to put in a fixed position of the stack. Also it provides external functions for collaboration of modules in the stack and management of cross-layer functionalities.

4.3 Implementation of the CLMM

The CLMM is implemented in NS-MIRACLE. The implementation consists of:

- An implementation of the MonitorPlugIn, tmCIMessage, Metrics, Mac_Interface and storage module;
- An extension and modification of NSMIRACLE PlugIn module, CIMessage module, Mac_802.11 module;
- A modification of Aodv-uu for NSMIRACLE [29];
- A modification of dei80211mr for NSMIRACLE;

- Port Garretto patch [28] for NS2. The Garretto patch [28] for NS-2 monitors MAC layer busy time. From the MAC busy time the MAC metrics can be derived.
- Two scenarios evaluate the CLMM and analyze the results.

The basic modules of CLMM are MonitorPlugIn, tmCIMessage, Metrics, Interface, Routing module. Figure 4-5 shows the relationship of classes of CLMM. In this section, we will describe these modules and their classes and functions in detail.

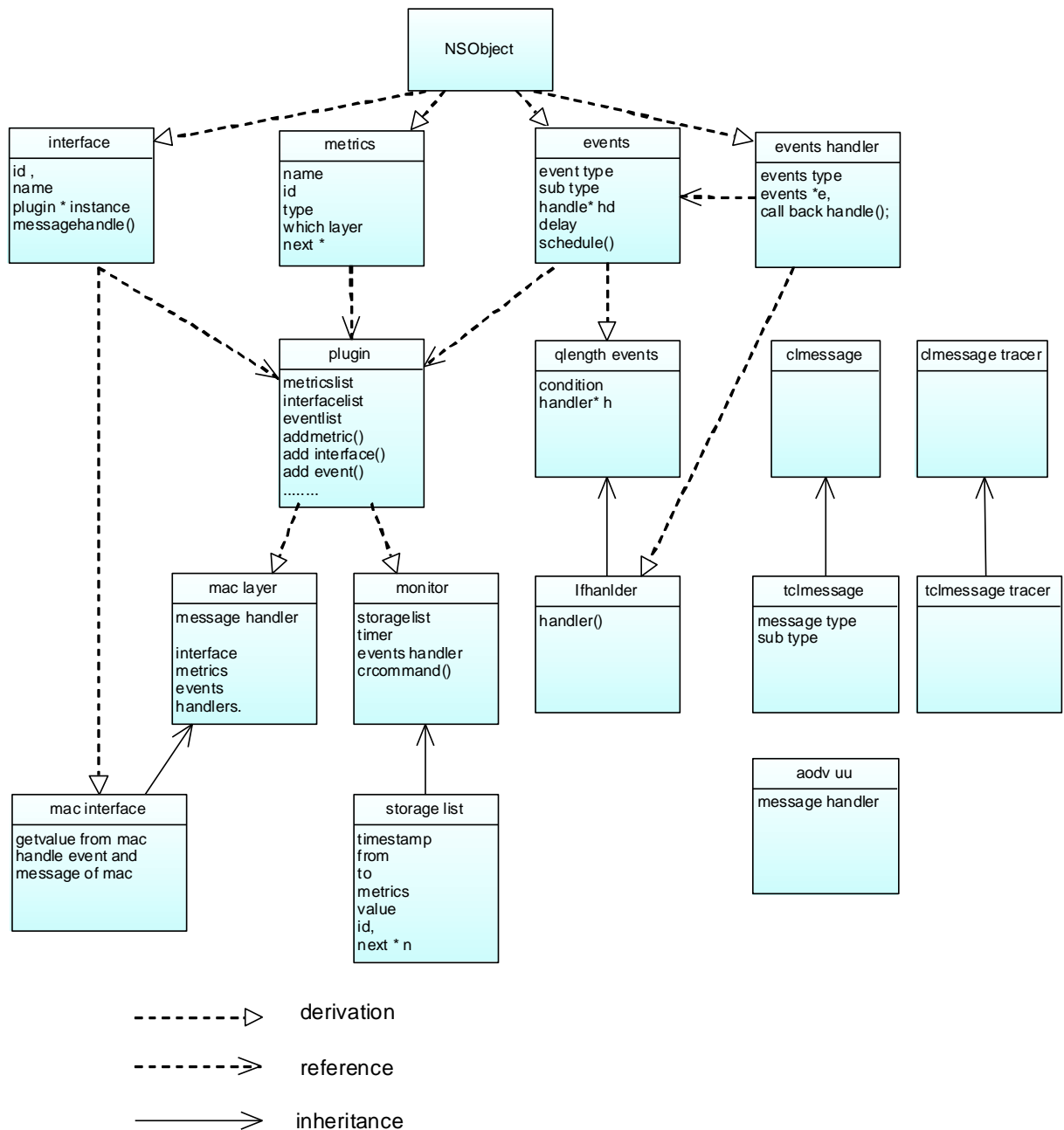


Figure 4-5 The Relationship of Classes of CLMM

4.3.1 The MonitorPlugIn Module

The MonitorPlugIn module defines the two monitoring mechanisms that provide time-driven and event-driven monitoring functions for cross-layer monitoring. It handles cross-layer messages receiving and delivering. It calculates the average values for these metrics. It also has a storage sub-function which stores the results in a data pool. Using these results, the MonitorPlugIn module not only outputs statistic data for the simulations and evaluations, but also supports other modules (i.e. the action module) to perform data retrieving and comparing functions for network performance optimization mechanisms.

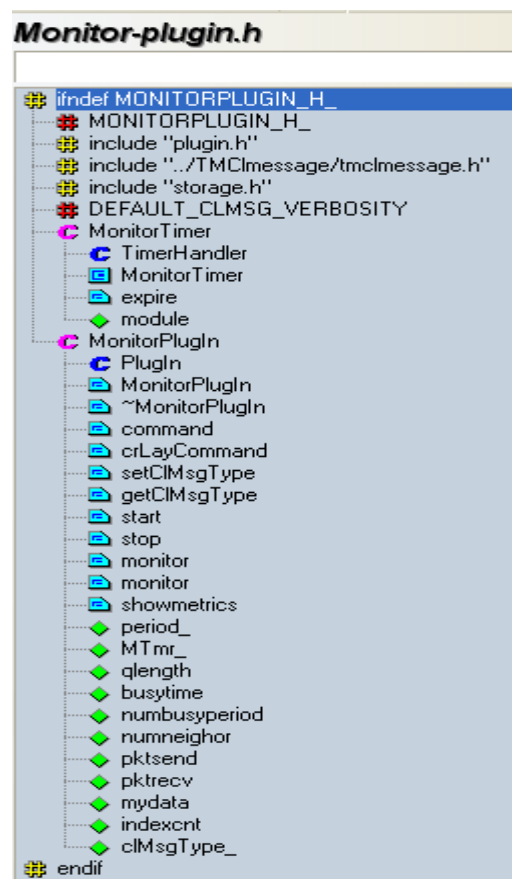


Figure 4-6 The structure of the MonitorPlugIn class

The structure of MonitorPlugIn module is shown in Figure 4-6. The MonitorPlugIn module is inherited from PlugIn class [19], which is the father class of Module in NSMIRACLE. As described in 4.2.1.3, the NSMIRACLE PlugIn Module has the message events handling function, message exchanging and delivering function. All modules are connected by dedicated objects, referred to here as Connectors [19]. The MonitorPlugIn class has a built-in

friend class MonitorTimer which carries out timing monitoring action. All the messages are collected via this module. The virtual function crLaycommand () handles the two kinds of the cross layer messages: INTERFACEMSG and ROUTINGMSG. When an event occurs during the run time, an 'onevent' message is sent to the monitorPlugIn module. The crLaycommand() handles the message by checking the event type and taking an action accordingly. For example, if a queue overflow event occurs, the monitorPlugIn sends out a signal message with the value of the metric. If a metric event (I.e. macpktsend or macpktrcv) occurs, the monitorPlugIn fetches the values of the metric, calculates the average values and stores the results to the storage module. When the monitorPlugIn receives a 'queryresult' message, it fetches the values of the metrics, calculates the average values and stores the results to the storage module.

Function Start () is used to broadcast messages to all modules and discover the modules that can be monitored. It is also used to send the message to these modules. The member Mtmr_ is a timer class which has timing function for the MonitorPlugIn. When the modules are ready, the Mtmr_ triggers monitor action by period.

```

Storage.h
-#ifndef STORAGE_H_
-#define STORAGE_H_
-#include <iostream>
-#include <ostream>
-#include <string>
-#include "../TMCmessage/tr
+ListNode
+ListNode
+ListNode
+ListNode
+index_
+et_
+nodename_
+value_
+timestamp_
+nextNode
+StorageList
+StorageList
+~StorageList
+StorageList
+operator=
+insert
+remove
+find
+output
+size
+length
+headNode
+currentNode
+recursiveNode
-#endif
  
```

Figure 4-7 The structure of the Storage Module

4.3.2 The Storage Module

The Storage module is a sub-module of the MonitorPlugIn module. The storage module stores all the values of metrics that the CLMM monitor. The structure of this module is shown in

Figure 4-7. The storage module has the basic functions of insert, remove, find and output operation.

4.3.3 The Mac_Interface Module

The Mac_Interface module is inherited from Interface Class. As showed in Figure 4-8, the MAC_interface module consists of recv(), crLayMsgHandle() and onEventAction() classes. The recv() class receives a cross-layer message during the run time. The crLayMsgHandle() class handles the message according to the type of the message. If the message type is MONITORMSG, it answers the monitorPlugIn module by sending back the INTERFACE message to the monitorPlugIn module. If the message is a query message, it fetches the values of the metrics and sends the results back to the monitorPlugIn module. The onEventAction() implements event-driven mechanisms for the CLMM. Currently this function handles two kinds of events: QLENGTHEVENT (a queue overflow event) and MACEVENT (a metric event, for example, a packet receive event). If the event is a QLENGTHEVENT event, which means a queue overflow event occurs, the Mac_interface module retrieves and updates the values of queue length. Then it sends out a signal message (an 'onevent' message) to the monitorPlguIn module. If the event is a MACEVENT, the Mac_interface module updates the value of the metric and sends a message to the monitorPlugIn module.

The Mac_Interface module is a module hooked to the monitorPlugIn modules. When it is loaded in Mac modules, it takes in charge of all message delivery, data collection, message processing and event handling. It also responses to the monitor module command.

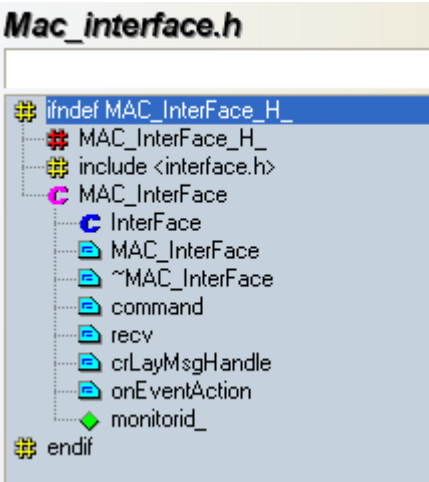
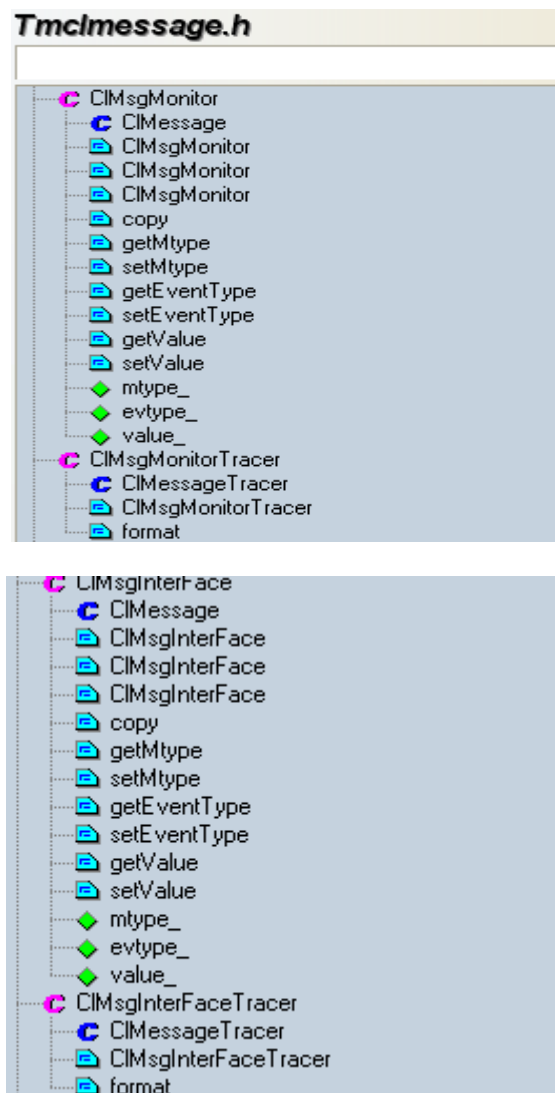


Figure 4-8 The Structure of the Mac_interface Module

4.3.4 The tmCIMessage Module

The tmCLMessage Module has two kinds of base classes: the CIMessage and CLMessage Tracer classes. As shown in Figure 4-9, this module consists of three message classes and tracer classes:

- CIMsgMonitor() and CIMsgMonitorTracer(),
- CIMsgInterface() and CIMsgInterfaceTracer(),
- CLMsgRouting() and CLMsgRoutingTracer().



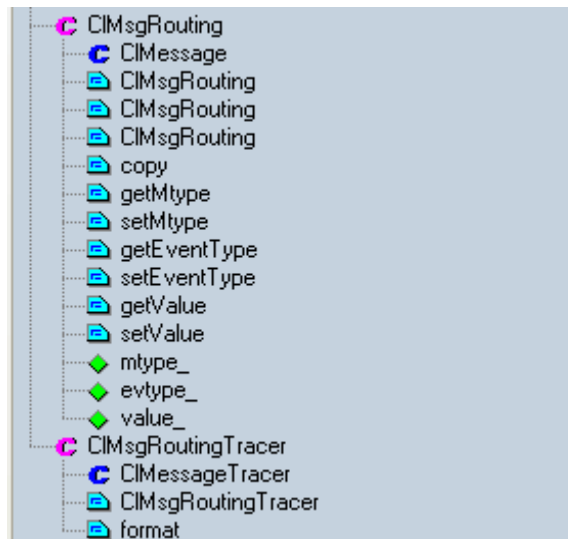


Figure 4-9 The Structure of tmCLMessage Module

The tmCLMessage module defines the basic messages, the message types and sub-types. Every message has its source id and destination id. The message types and sub-types for the CLMM are shown in Table3-3 and Table3-4. By using this module we can define new message types and sub-message types. We can also derive new messages and message handle functions based on the existing messages.

The CIMsgMonitor(), CIMsgInterface() and CLMsgRouting() classes are derived from CLMessage class. The basic message handle functions for these class are: getMtype(), setMtype(), getEventType(), setEventType(), getValue(), setValue(). Furthermore, these classes have their own message handle functions.

The CIMsgMonitorTracer(), CIMsgInterfaceTracer() and CLMsgRoutingTracer() are derived from CLMessageTracer class. With these ClmessageTracer classes, the message route module is implemented. The tmCLMessage module checks the source id and destination id and delivers the messages to the destination.

4.3.5 Metrics module

The Metrics class is inherited from NSObject class. The Metrics class provides a set of functions to support the operations of metrics during the run time. With these function, each metric can be set up and get by its name, type and ID number.

4.3.6 Routing module

We modified the Aodv-uu for the NSMIRACLE [29] for the CLMM Routing module. The routing module not only provides routing function for the CLMM, but also supports message delivery for the CLMM. Furthermore, with using the Aodv-uu for the NSMIRACLE, the routing module also provides routing metrics (for example, the number of one hop neighbors) for the CLMM.

4.3.7 Ifhandler class

Ifhandler class handles events for the layer interfaces when the events occur. Two kinds of events are implemented for this class: QLENGTHEVENT and MACEVENT. The function is shown in Figure 4-10.

```
void
Ifhandler::handle(Event *e)
{
    if(((Ifevent*)e)->etype_==QLENGTHEVENT)
    { ((Interface*)this->parent_)->onEventAction(((Ifevent*)e)->etype_);
      }
    if(((Ifevent*)e)->etype_==MACEVENT)
    { ((Interface*)this->parent_)->onEventAction(((Ifevent*)e)->etype_);
      }
};
```

Figure 4-10 The Ifhandler Function

4.3.8 Interface class

Interface class supports a new interface setup and retrieving. It has a set of functions to set up a new interface for the CLMM. Each interface has its name and id.

4.4 Otcl Script

To test the functionality of the CLMM module and analyze the simulations, two scenarios were written by Otcl. To monitor the cross-layer messages, some modules are added in the script according to the implementation the CLMM and modification of NSMIRACLE modules.

MonitorPlugIn

The command of setting up a new PlugIn module is shown as below:

```
set plg [new MonitorPlugIn]
```

When the MonitorPlugIn has been set up, it can be configured by adjusting the monitoring period value, the command is shown as below:

```
$plg set period_ value
```

To attach the MonitorPlugIn to the nodes, we use the following command:

```
set p [$node addPlugin $plg 1 "PLUGIN"]
```

MAC_Interface Module

To add a new MAC_Interface module for the CLMM, we use the command as below,

```
set macinf [new MAC_InterFace]
```

Once the MAC_Interface module is set up, the MAC_Interface can be configured by setting up thresholds of the metrics, id and name. The configuration commands are:

```
$LL set Qlen_threshold value
```

```
$macinf setid value
```

```
$macinf setname "string"
```

To attach a new MAC_interface module to the link layer, we use the following command:

```
$LL($n) addInterface $macinf
```

Metrics module

The different metrics are set up by using following command:

```
set met [new QlenMetric]
```

Each metric is configured by its ID and name:

```
$met setid value
```

```
$met setname "string"
```

The new metric can be added to the link layer by using a command:

```
$LL($n) addMetrics $met
```

The Variable of Metrics

Six variables are set up in the scripts. The variables are shown in Table4-9. To get the values of the metrics during the run time, we use following commands:

```
set bt($i) [$plg($i) set busytime_ave]
```

```
set ql($i) [$plg($i) set qlength_ave]
```

```
set num_bp($i) [$plg($i) set num_busyperiod_]
```



```

set num_neighbor($i) [$plg($i) set num_neighbor_]
set send($i) [$plg($i) set pktsend_ave]
set rcv($i) [$plg($i) set pktrecv_ave]

```

variable	Description
busytime_ave	the average busytime
qlength_ave	The average queue length
num_busyperiod_	the number of busy period
num_neighbor_	the number of one hop neighbors
pktsend_ave	the average number of packets send per second
pktrecv_ave	the average number of packets received per second

Table 4-1 Variables of the Metrics

Module/AODVUU

The Module/AODVUU is attached to each node. The setup command is:

```
$set Aodvs [new Module/AODVUU]
```

The command of setting up IP address and subnet for the each node is:

```

$set IpIfs [new Module/IP/AODVInterface]
$IpIfs addr value
$IpIfs subnet value

```

Output Stored Data

In order to verify the results of the stored data, we use a command to output all the data that the CLMM monitored:

```
$ns at end_time "$plg($n) showmetrics"
```

Start/Stop commands

To start and stop AODVUU and PlugIn during the run time, we use the “start” and “stop” commands in the scripts:

```

Aodv-uu start
Aodv-uu stop
PlugIn start
PlugIn stop

```

Two simulation scripts named clmm_mr.tcl and clmm_miracle.tcl are used to verify the performance of the CLMM module. Each simulation uses xgraph tool to draw the graph of the metrics. The graphs of metrics are aggregated into one aggregated graph.

5 Simulation and Evaluation

5.1 Simulation setting

Two simulations are written for the CLMM. The main goal of simulation is to evaluate the CLMM. In the first simulation we use NSMIRACLE MAC/802_11/Multirate for MAC layer. In the second simulation we use MAC/802_11/Miracle for MAC layer. The difference between MAC/802_11/Multirate and MAC/802_11/Miracle is link character. MAC/802_11/Multirate uses multi-rate MAC while MAC/802_11/Miracle uses single-rate for simulation. By default NSMIRACLE uses MAC/802_11/Multirate. To evaluate CLMM and compare the result of simulation, we modify MAC/802_11/Miracle MAC and make it usable for CLMM. Each simulation generates two CBR traffics. The first CBR traffic (CBR1) is sent from node0 to node4. The second CBR traffic (CBR2) is generated between node5 and node6. The first CBR traffic starts at the 1st second. The second CBR traffic begins at the 31st second. The sending rate of the second CBR increases 100 packets every 10 seconds during the first 190th seconds. The sending rate of the second CBR decreases 100 packets every 10 seconds after the 200th seconds. Because there is no variable “rate_” in NSMIRACLE, we can only set up CBR rate by using interval variable “period_”. The setting of the simulations is shown in Table5-1. Each simulation runs two times. The first time we use topology 1. The second time we use topology 2. The difference between these two topologies is the distance of the wireless mesh nodes.

Name	Value
simulation time	200
PHY LAYER	
Carrier Sensing threshold	1.47E-011
Receiving power threshold	7.14E-011
Transmitter signal power	7.21E-003
Frequency	2.47E+009
Propagation Model	Freespace
Path loss	1
MAC LAYER	
RTS/CTS	off
Routing	
Protocol	Aodv-uu*
Application1	CBR1
Packet size	1000Byte
Period	0.01

CBR1 start time	1st second
CBR1 end time	291 th second
Application1	CBR2
Packet size	1000Byte
Period	*
CBR2 start time	31 th second
CBR2 end time	261 th second
* Aodv-uu for NSMIRACLE [29]	

Table 5-1 Simulation Setting

5.2 Simulation Topology

The simulation topologies are shown in Figure 5-1 and Figure 5-2. There are 7 wireless mesh nodes in each scenario. Node1, node2 and node3 are the nodes that only forward the traffic. The distance between each one hop nodes in topology one is 160m. In topology two, the distance between node5 and node 1, node6 and node2 are 300m. The difference distance should result different values of the metrics due to the nature of interference in wireless mesh networks.

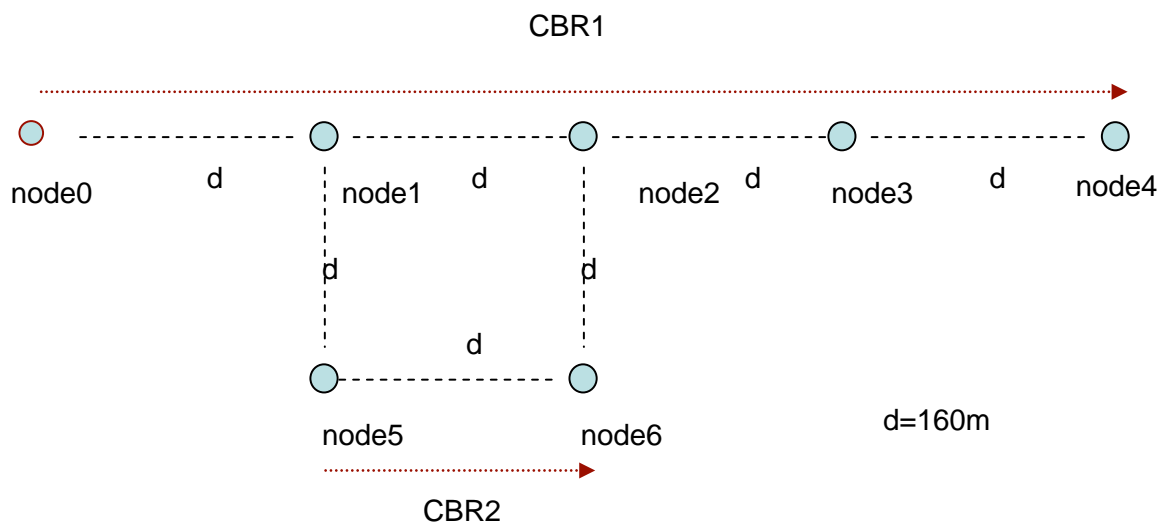


Figure 5-1 Topology One

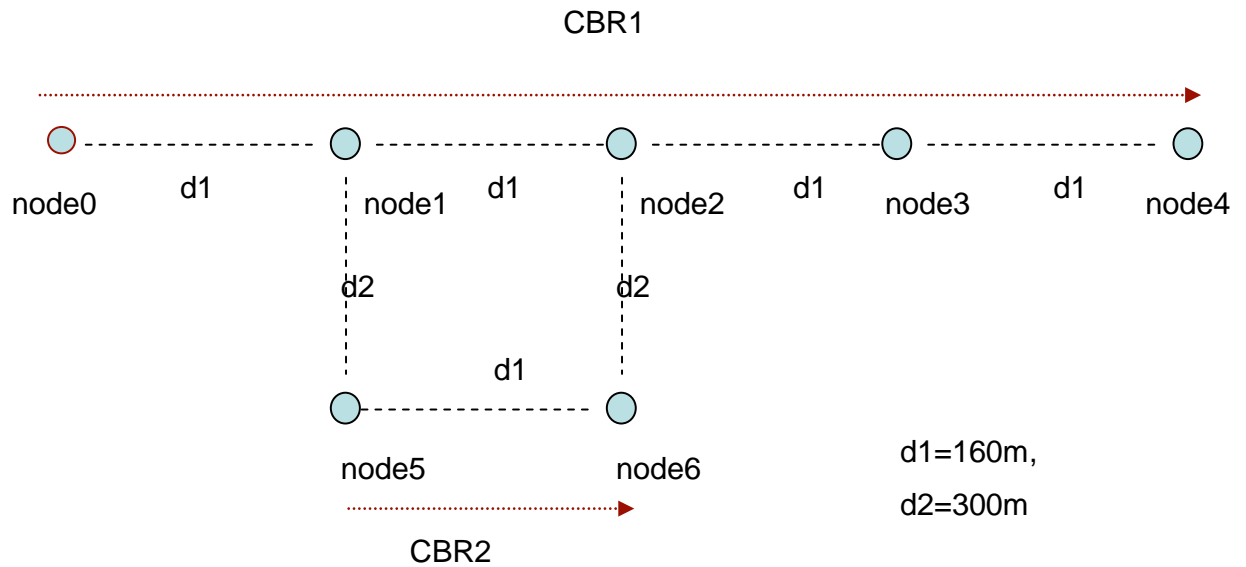


Figure 5-2 Topology Two

5.3 Simulation Results

The simulation 1 evaluates the performance of CLMM using NSMIRACLE MAC/802_11/Multirate. The results are shown in Figure 5-3--Figure 5-10. We will discuss these results in following section.

5.3.1 Cross-layer messages

The cross-layer messages are traced and output to the cltrace.tr file. As shown in Figure 5-3, the first column is the type of module, the second is the simulation time, the next one is node ID, the fourth column is the source module name, the fifth is the destination name, the sixth column includes message id, source id, destination type (unicast or broadcast) and destination id, the last column is the name of cross-layer message.

```

N 17.000000000 5 PLUGIN(5) NC ASYNC [CMN: 10391 6 BRD -1 N] monitorplugin: monitor message ]
P 17.000000000 5 NC PHY ASYNC [CMN: 10391 6 BRD -1 M] monitorplugin: monitor message ]
P 17.000000000 5 NC LL ASYNC [CMN: 10391 6 BRD -1 M] monitorplugin: monitor message ]
N 17.000000000 5 LL NC ASYNC [CMN: 10392 4 UNI 6 N] Interface: broadcast message ]
P 17.000000000 5 NC PLUGIN(5) ASYNC [CMN: 10392 4 UNI 6 M] Interface: broadcast message ]
N 17.000000000 5 LL NC ASYNC [CMN: 10393 4 UNI 6 N] Interface: broadcast message ]
P 17.000000000 5 NC PLUGIN(5) ASYNC [CMN: 10393 4 UNI 6 M] Interface: broadcast message ]
N 17.000000000 5 LL NC ASYNC [CMN: 10394 4 UNI 6 N] Interface: broadcast message ]
P 17.000000000 5 NC PLUGIN(5) ASYNC [CMN: 10394 4 UNI 6 M] Interface: broadcast message ]
P 17.000000000 5 NC IP ASYNC [CMN: 10391 6 BRD -1 M] monitorplugin: monitor message ]
P 17.000000000 5 NC UU ASYNC [CMN: 10391 6 BRD -1 M] monitorplugin: monitor message ]
P 17.000000000 5 NC TRANS ASYNC [CMN: 10391 6 BRD -1 M] monitorplugin: monitor message ]
P 17.000000000 5 NC APP_CBR ASYNC [CMN: 10391 6 BRD -1 M] monitorplugin: monitor message ]

```

Figure 5-3 The Cross Layer Message Trace File

When a layer event occurs, the CLMM sends out a layer event message and updates the values of the metrics (shown in Figure 5-4).

```

PLUGIN(1): receive a message from MACinterface : packets send per sec: --=> 12.000000
PLUGIN(1): layer events occurred.
PLUGIN(1): receive a message from MACinterface : packets receive per sec: --=> 47.000000

```

Figure 5-4 A Layer Event Message

When a threshold of a metrics is exceeded during the run time, the CLMM also reacts by send out a signal message. Figure 5-5 shows this message sent by CLMM when the queue length reaches its threshold.

```

MAC layer : qlength overflow events had occurred.....
PLUGIN(5): layer events occurred.
EVENT:queue length over threshold. --=> 99.000000
 macinterface(5):successful on update metrics.
PLUGIN(5): layer events occurred.
EVENT:queue length over threshold. --=> 99.000000
 macinterface(5):successful on update metrics.
MAC layer : qlength overflow events had occurred.....
PLUGIN(5): layer events occurred.
EVENT:queue length over threshold. --=> 99.000000
 macinterface(5):successful on update metrics.

```

Figure 5-5 A Queue Length Overflow Event Message

5.3.2 Results from Storage Module

To show the values stored in the storage module, we use the following command in the Otel Scripts:

\$ns at end_time “plg\$ns showmetrics”

The result is shown in Figure 5-6.

```
Index: 256 Event_Type: 0 Metrics Name: queuelength Value: 0 TimeStamp: 3.9
Index: 257 Event_Type: 5 Metrics Name: num of busy period Value: 873 TimeStamp: 3.9
Index: 258 Event_Type: 4 Metrics Name: mac busytime Value: 0 TimeStamp: 3.9
Index: 259 Event_Type: 0 Metrics Name: queuelength Value: 0 TimeStamp: 3.9
Index: 260 Event_Type: 5 Metrics Name: num of busy period Value: 873 TimeStamp: 3.9
Index: 261 Event_Type: 4 Metrics Name: mac busytime Value: 0 TimeStamp: 3.9
Index: 262 Event_Type: 8 Metrics Name: num of neighbor Value: 2 TimeStamp: 3.9
Index: 263 Event_Type: 2 Metrics Name: packets send per sec Value: 100 TimeStamp: 4
Index: 264 Event_Type: 3 Metrics Name: packet rcv per sec Value: 300 TimeStamp: 4
Index: 265 Event_Type: 0 Metrics Name: queuelength Value: 0 TimeStamp: 4
Index: 266 Event_Type: 5 Metrics Name: num of busy period Value: 903 TimeStamp: 4
Index: 267 Event_Type: 4 Metrics Name: mac busytime Value: 0 TimeStamp: 4
Index: 268 Event_Type: 0 Metrics Name: queuelength Value: 0 TimeStamp: 4
Index: 269 Event_Type: 5 Metrics Name: num of busy period Value: 903 TimeStamp: 4
Index: 270 Event_Type: 4 Metrics Name: mac busytime Value: 0 TimeStamp: 4
Index: 271 Event_Type: 8 Metrics Name: num of neighbor Value: 2 TimeStamp: 4
Index: 272 Event_Type: 0 Metrics Name: queuelength Value: 0 TimeStamp: 4.1
Index: 273 Event_Type: 5 Metrics Name: num of busy period Value: 933 TimeStamp: 4.1
Index: 274 Event_Type: 4 Metrics Name: mac busytime Value: 0 TimeStamp: 4.1
Index: 275 Event_Type: 0 Metrics Name: queuelength Value: 0 TimeStamp: 4.1
Index: 276 Event_Type: 5 Metrics Name: num of busy period Value: 933 TimeStamp: 4.1
Index: 277 Event_Type: 4 Metrics Name: mac busytime Value: 0 TimeStamp: 4.1
Index: 278 Event_Type: 0 Metrics Name: queuelength Value: 0 TimeStamp: 4.2
Index: 279 Event_Type: 5 Metrics Name: num of busy period Value: 963 TimeStamp: 4.2
Index: 280 Event_Type: 4 Metrics Name: mac busytime Value: 0 TimeStamp: 4.2
Index: 281 Event_Type: 0 Metrics Name: queuelength Value: 0 TimeStamp: 4.2
Index: 282 Event_Type: 5 Metrics Name: num of busy period Value: 963 TimeStamp: 4.2
Index: 283 Event_Type: 4 Metrics Name: mac busytime Value: 0 TimeStamp: 4.2
Index: 284 Event_Type: 0 Metrics Name: queuelength Value: 0 TimeStamp: 4.3
Index: 285 Event_Type: 5 Metrics Name: num of busy period Value: 993 TimeStamp: 4.3
Index: 286 Event_Type: 4 Metrics Name: mac busytime Value: 0 TimeStamp: 4.3
```

Figure 5-6 The Output of Storage Module

5.3.3 Metrics

5.3.3.1 Simulation 1 with Topology 1

The Number of Packets Sent per Second

The number of packets sent per second of node5 increases 100 packets every 10 seconds at the first 150 seconds. After that, when node5 tries to send more data than the MAC layer and channel speed can handle, the packets will queue up in the packet queue. When the packets rate reaches the maximum MAC service rate, this will lead to a queue overflow. New packets can not be stored in the queue and dropped. After the 201th second, the sending rate of CBR2 drops 100 packets every 10 seconds. Because the sending rate of CBR1 does not change during the run time, the number of packets sent per second of node0 stays at one level accordingly.

The Number of Received Packets per Second

Figure 5-7 shows that the number of received packets per second of node6 changes according to the dynamic sending rate of node5. When the sending rate of node5 reaches the maximum MAC service rate, packet loss occurs. The number of received packets per second of node6 stays at a high level instead of increasing with the sending rate. After the 241th second, when the queue length drops with the decreasing sending rate of node5, the number of packets received changes with sending rate again. The number of received packets per second of node4 stays at one level due to one sending rate of CBR1. Since each node can only communicate with its one hop neighbors in topology 1, the number of received packets per second of node1 and node2 also follows the changes with node5 and node6.

The Queue Length

We can also find the relationship between the number of sent packets per second and the queue length. If node5 tries to send more data than the MAC layer and channel speed can handle, the packets will queue up in the packet queue. From the 151th second to the 241th second the queue length of node5 jumps sharply and stays at a high level. When the packets rate reaches the maximum MAC service rate, this will lead to a queue overflow.

The Busy Time and the Number of Busy Period

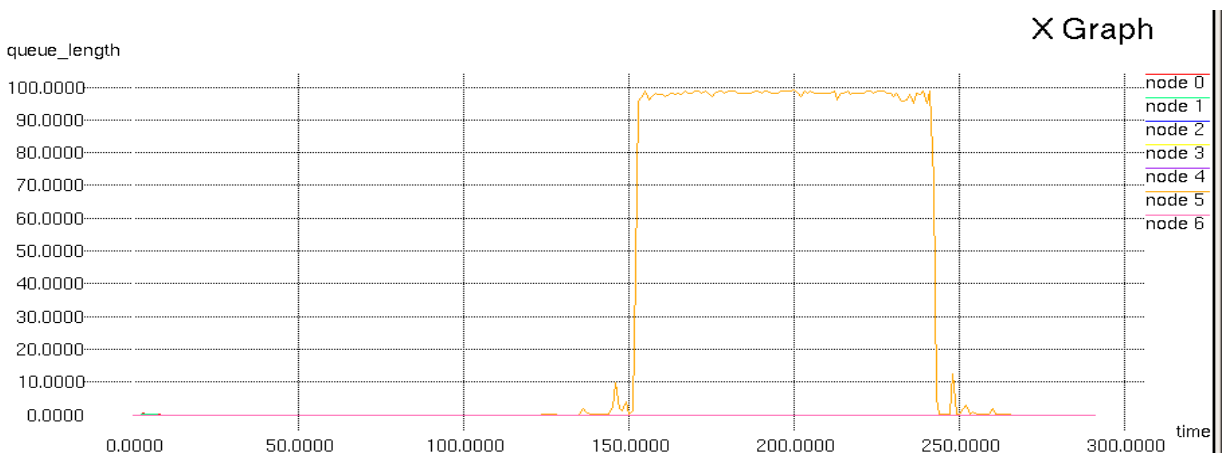
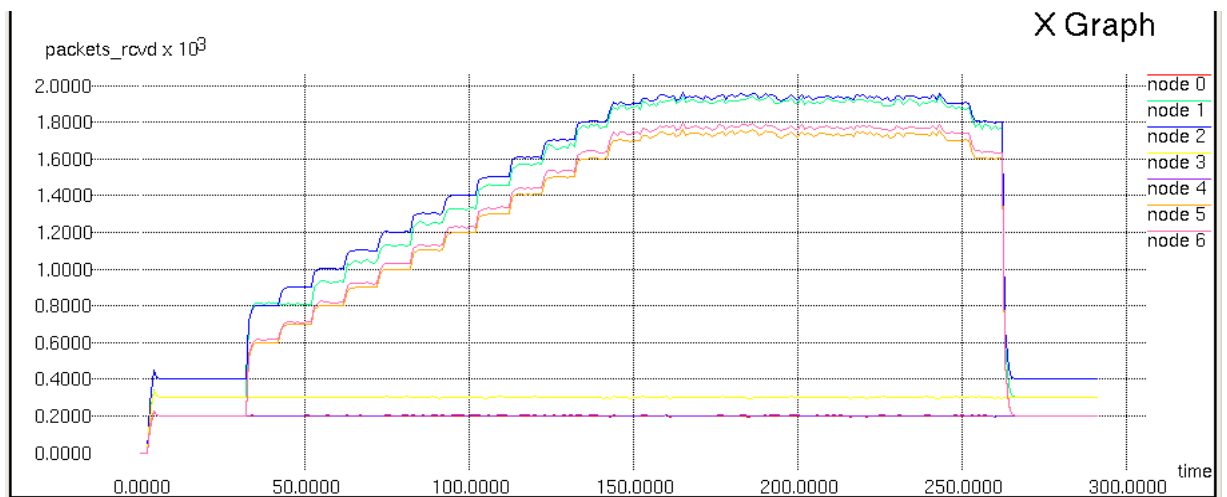
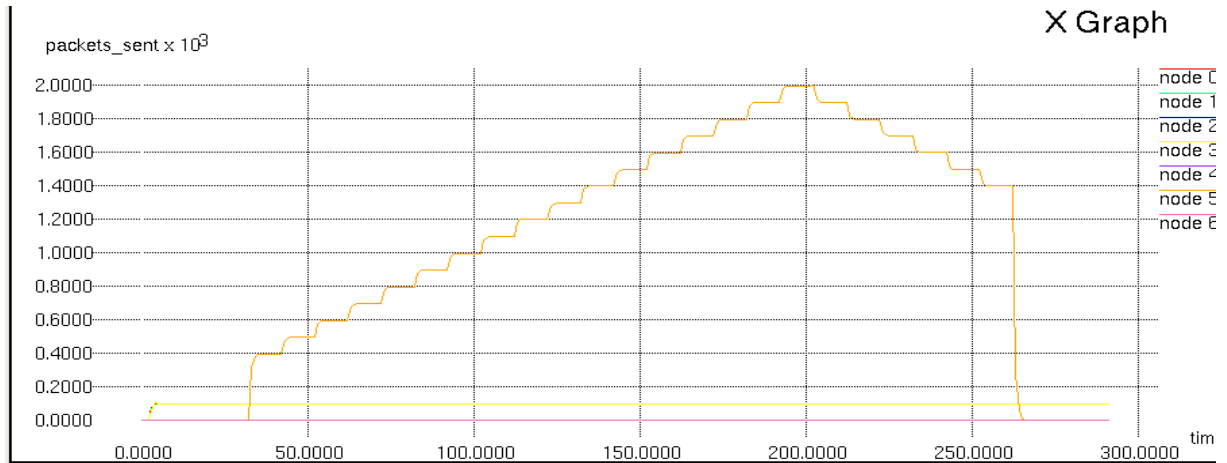
In topology 1, since each node can communicate with its one hop neighbors, the number of busy period of node1, node2 and node6 rises quickly due to the rising of the packets sent per second of node5. And because node1 is between node0 and node5, node1 can receive more packets from node0 and node5, its busy time is higher than node6.

The number of busy period of node1, node2 and node6 increases with the rising packets sent rate of node5 too. The number of busy period of node2 is higher than node6. The number of busy period stays at a same level when MAC is not busy.

The Number of One Hop Neighbors

Figure 5-7 also shows the change of the number of one hop neighbors. When Aodv-uu starts, as each node can only communicate with its one hop neighbors in topology 1, every node gets the right number of its one hop neighbors. After 20 seconds, since there is no traffic between node5 and node6, the number of one hop neighbors of node1, node2, node5 and node6 drop one. We find if there is no traffic after Aodv-uu started, the number of neighbors always drops after 20 seconds. When the CBR2 starts at the 31th second, the number of one hop neighbors increases one. To some nodes, for example node1 and node2, the number of one hop neighbors of nodes drops again after the 51th second. The reason is the distance between node1, node5 and node2, node6 lead to interference between these nodes. The results of simulation1 with topology2 prove above assumption. When the distance between node1 and

node5, node2 and node6 are out of the sensing range, the number of one hop neighbors will be changed accordingly.



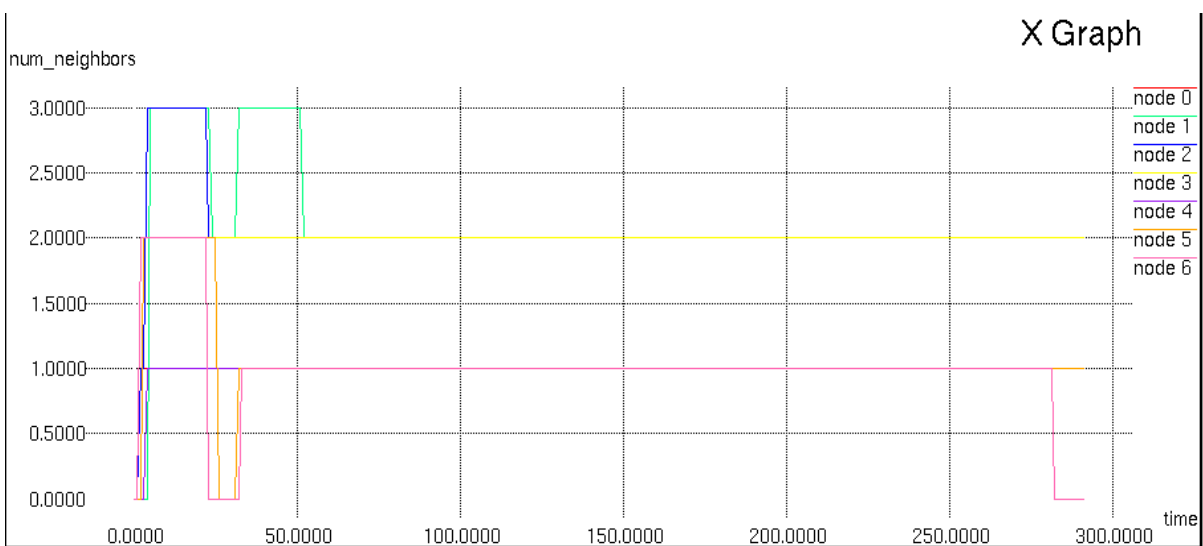
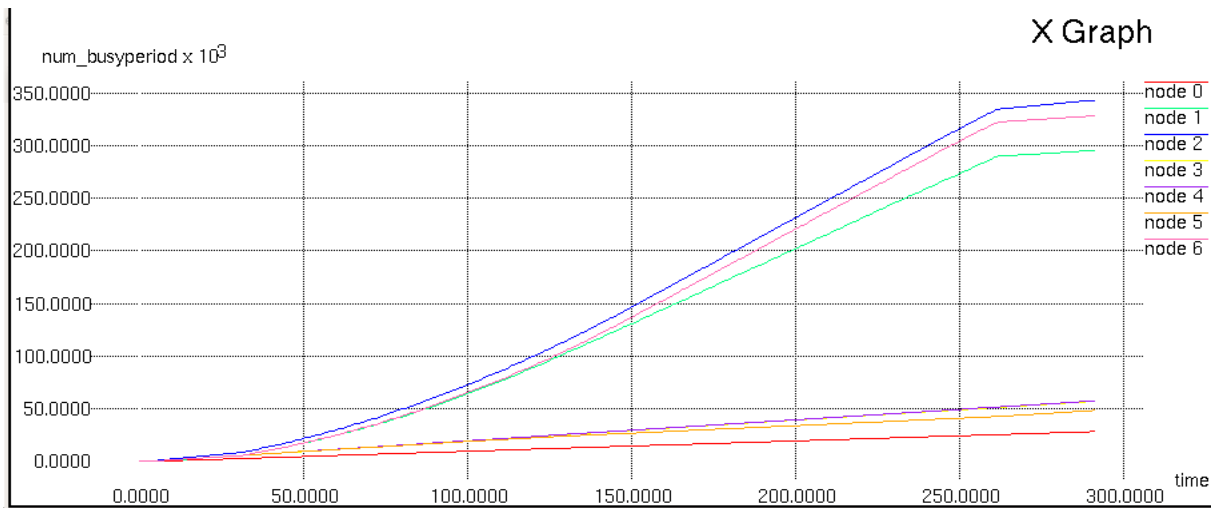
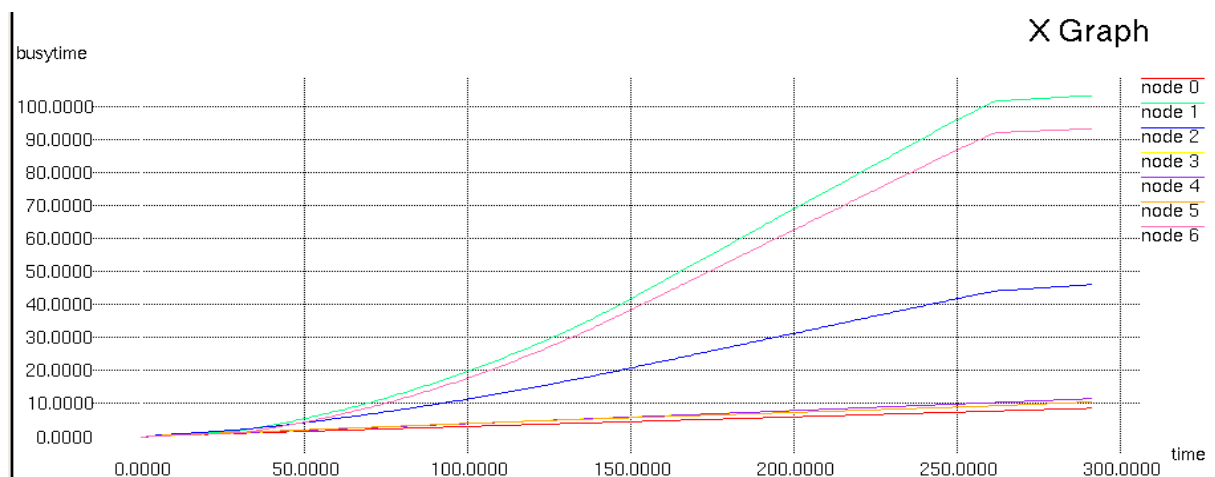


Figure 5-7 The Values of the Metrics (number of packets sent per second, number of packets received per second, busy time, number of one hop neighbors, number of busy period, queue length) in simulation1 using topology 1

5.3.3.2 The Simulation1 with Topology2

The Number of Packets Sent per Second

The number of packets sent per second of node5 increases 100 packets every 10 seconds at the first 160 seconds. From the 161th second to the 201th second, when node5 tries to send more data than the MAC layer and channel speed can handle, the packets will queue up in the packet queue. When the packets rate reaches the maximum MAC service rate, this will lead to a queue overflow. New packets can not be stored in the queue and dropped. After the 201th second, the sending rate of CBR2 drops 100 packets every 10 seconds. Because the sending rate of CBR1 does not change during the run time, the number of packets sent per second of node0 stays at one level accordingly.

The Number of Received Packets per Second

Figure 5-8 shows that the number of received packets per second of node6 changes according the dynamic sending rate of node5. When the sending rate of node5 reaches the maximum MAC service rate, packet loss occurs. The number of received packets per second of node6 stays at a high level instead of increasing with the sending rate. After the 231th second, when the queue length drops with the decreasing sending rate of node5, the number of packets received changes with sending rate again. The number of received packets per second of node4 stays at one level due to one sending rate of CBR1. Since node1 can not communicate with node5 and node2 can not communicate with node6, the numbers of received packets per second of node1 and node2 do not follow the changes with node5 anymore.

The Queue Length

Because of less interference between node1, node5, and node2 and node6, queue length takes longer time to reach its highest level and the time that queue length stays at high level is also shorter than the value in topology1. The queue reaches the highest level at the 161th second and drops after the 231th second.

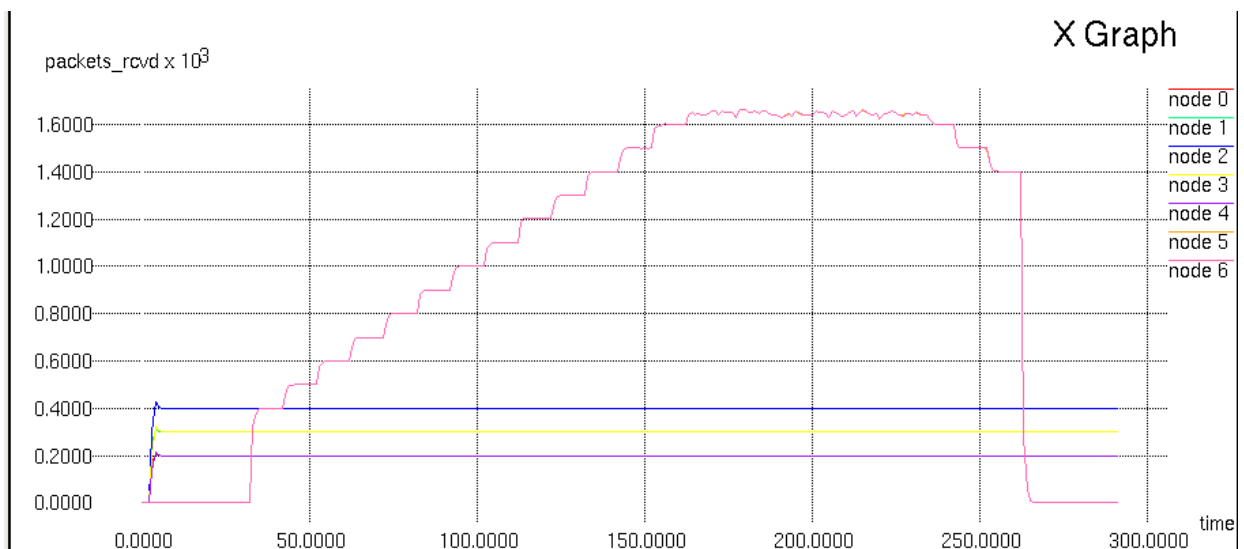
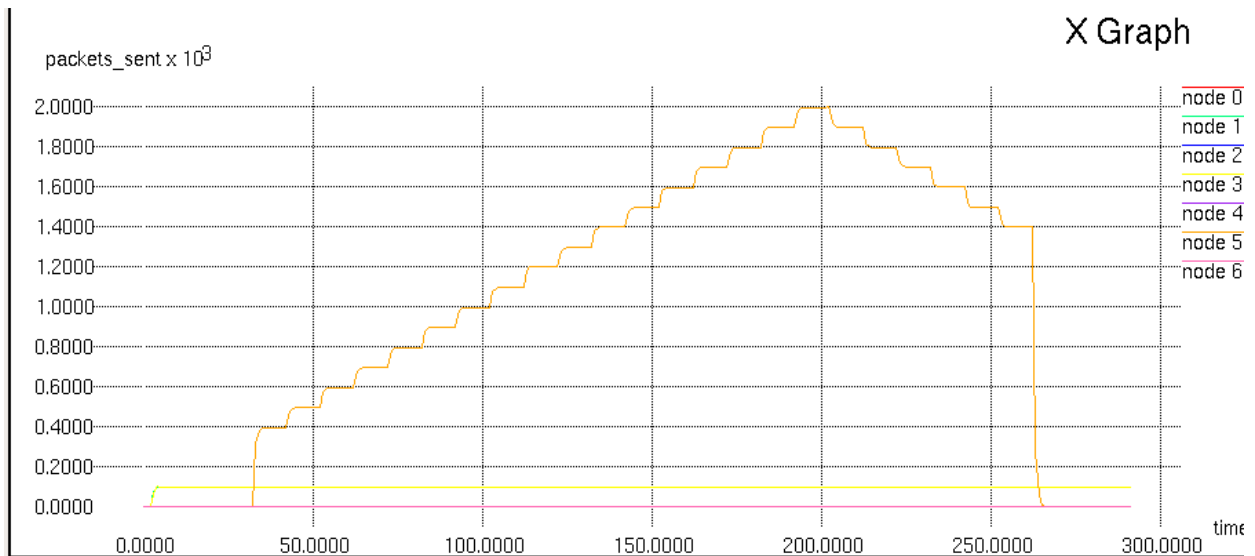
The Busy Time and the Number of Busy Period

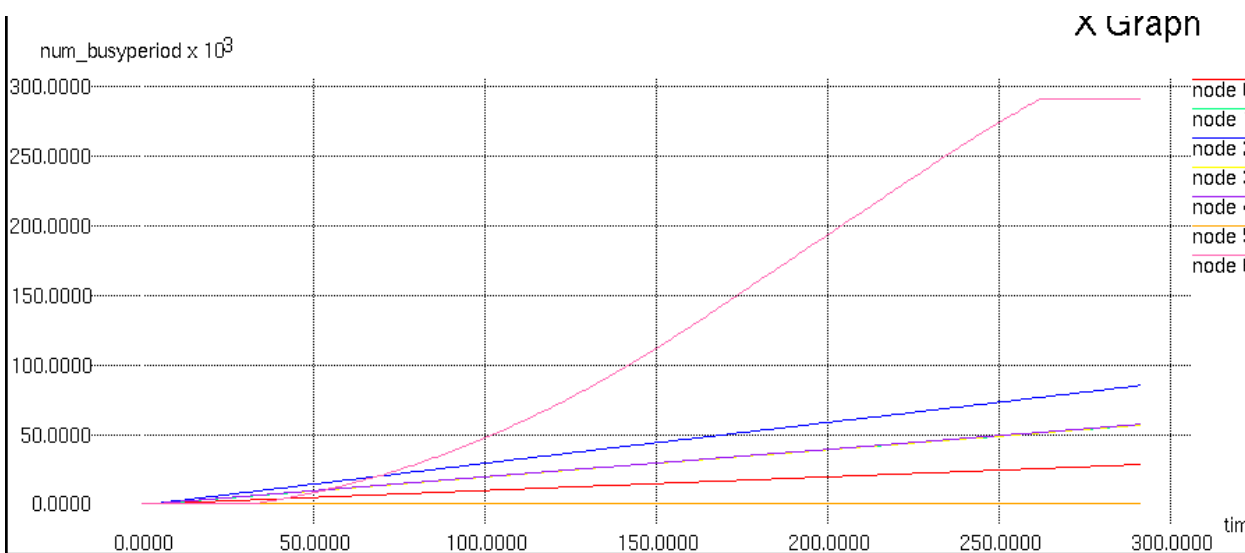
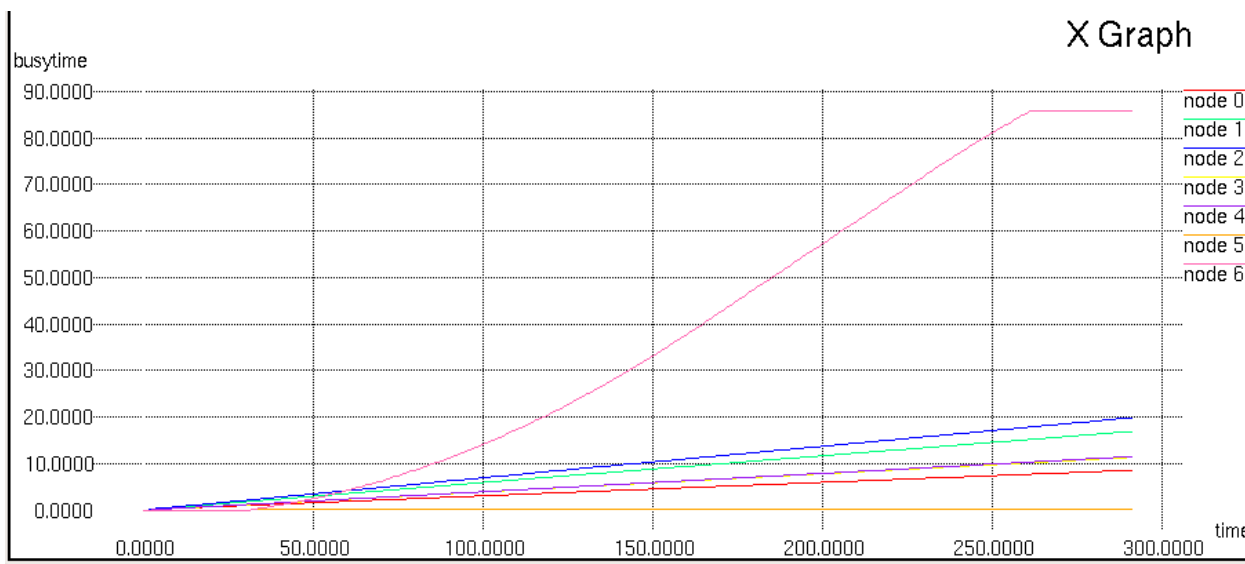
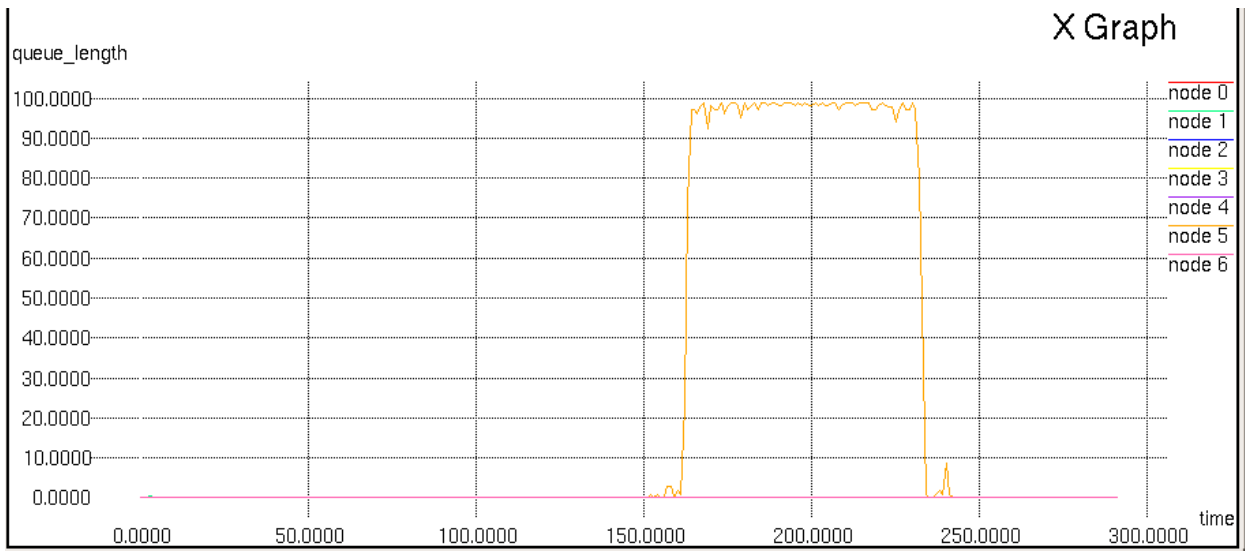
The busy time of node6 rises quickly due to the rising sending rate of node5. The number of busy period of node6 increases with the rising sending rate of node5 too. Since node1 can not communicate with node5 and node2 can not communicate with node6, the busy time and number of busy period of node1 and node2 is much lower than node6.

The Number of One Hop Neighbors

Figure 5-8 also shows the change of the number of one hop neighbors. Because node1 can not communicate with node5 and node2 can not communicate with node6, when Aodv-uu and

CBR1 starts, only node1, node2, node3 and node4 gets the number of its one hop neighbors. Node5 and node6 can not get the number of its one hop neighbors because there is no CBR2 traffic at the first 30 seconds. When the CBR2 is started at the 31th second, the number of one hop neighbors of node5, node6 increase one. Because of less interference between node1, node5, and node2 and node6, the number of one hop neighbors does not change after Aodv-uu and CBR traffic start.





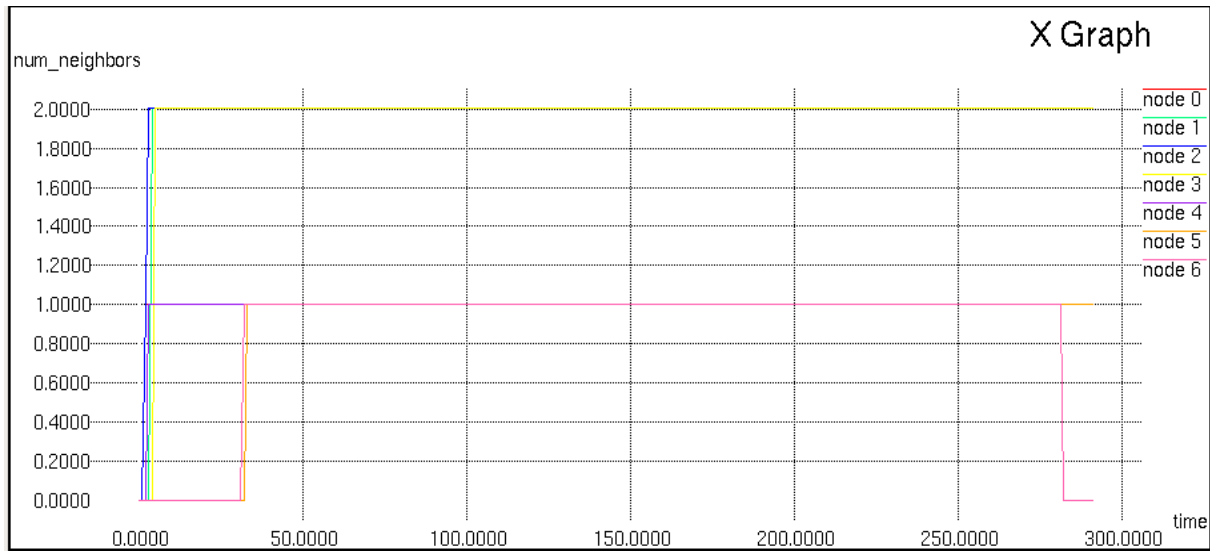


Figure 5-8 The Values of the Metrics (number of packets sent per second, number of packets received per second, busytime, number of one hop neighbors, number of busyperiod, queue length) in simulation1 using topology 2

5.3.3.3 Simulation2 with Topology1

We also evaluate the performance of CLMM with MAC/802_11/Miracle which uses a single rate for MAC. Simulation2 uses same two topologies described in section 5.2. The results of the simulation2 with topology1 are shown in Figure 5-9.

The Number of Packets Sent per Second

The number of packets sent per second of node5 increases 100 packets every 10 seconds at the first 190 seconds, and then decreases 100 packets every 10 seconds from the 200th second to the 260th second. The number of packets sent per second of node1, node2 and node3 is changing continuously during the simulation.

The Number of Received Packets per Second

The results of the number of received packets per second are quite different. The number of received packets per second of node6 does not rise with the sending rate accordingly. Instead these values stay at a low level. The maximum value is 108 packets per second. Because node1 can communicate with node5 and node2 can communicate with node6, when the CBR2 traffic starts, the number of received packets per second of node 3 and node4 drops sharply due to the interference. And they stay at a low level until the CBR2 stops.

The Queue Length

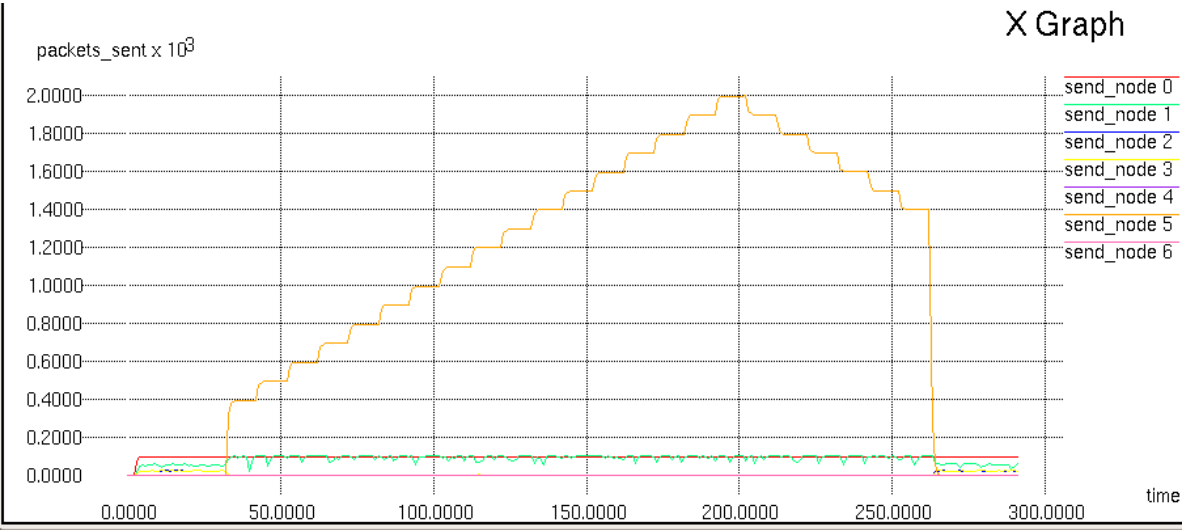
Due to the single MAC rate, the values of queue length of node0, node1 and node5 reach the highest level after the CBR traffics starts. After the CBR2 starts, the queue length of node0 changes greatly due to the interference. The interference degrades the link quality between node1 and node2. Because much less packets can be received, the queue length of node2 drops to zero after the 31th second.

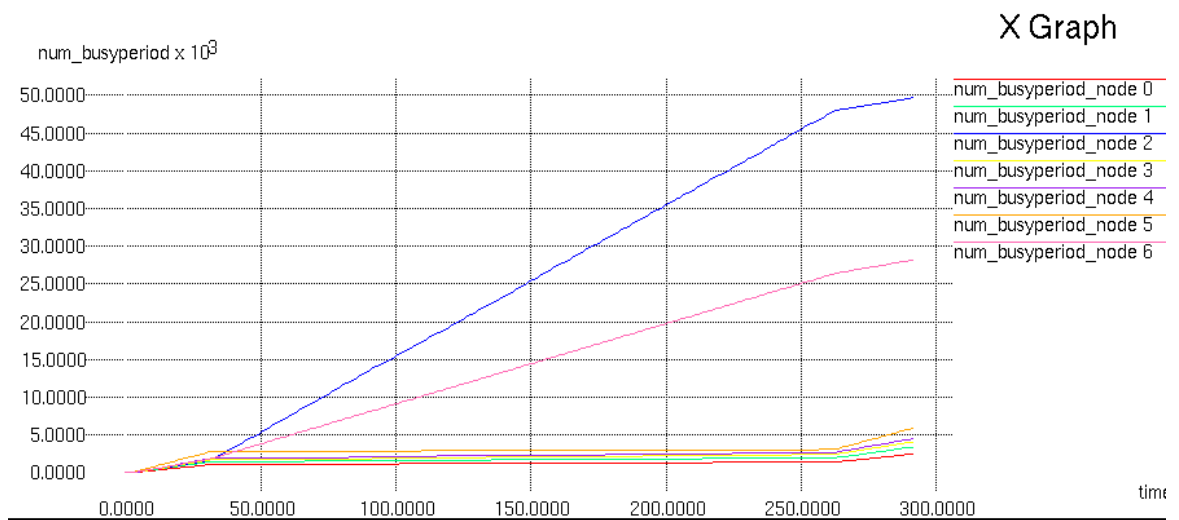
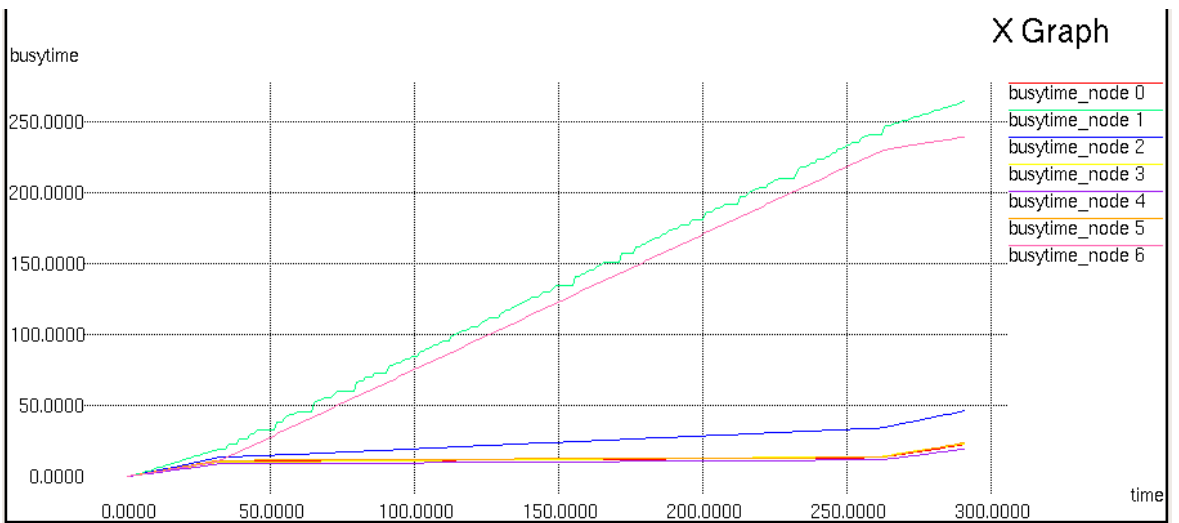
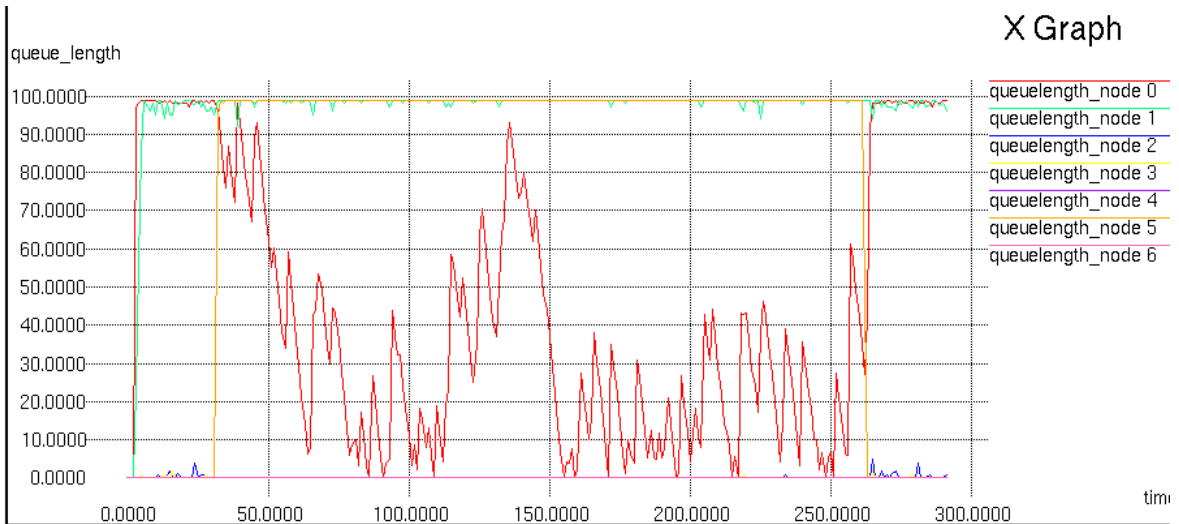
The Busy Time and the Number of Busy Period

The number of busy period of node1 and node6 rises quickly due to the rising of the packets sent per second of node5. And these values are much larger than the values in section 5.3.3.1. The number of busy period of node6 increases with the rising sending rate of node5 too. And number of busy period of node2 is higher than node6.

The Number of One Hop Neighbors

The change of the number of one hop neighbors is similar to section 5.3.3.1.





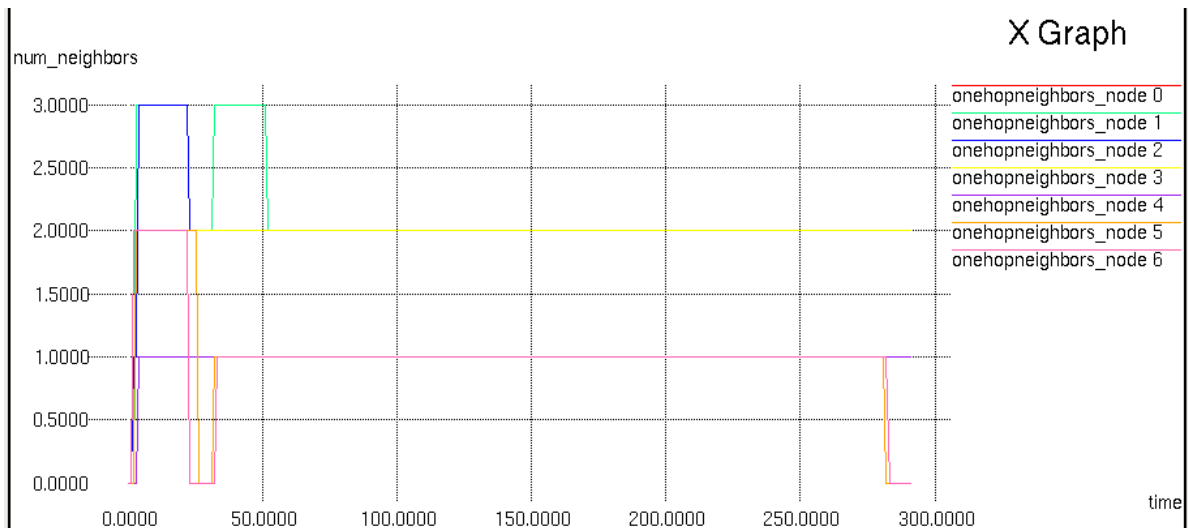


Figure 5-9 The Values of the Metrics (number of packets sent per second, number of packets received per second, busytime, number of one hop neighbors, number of busyperiod, queue length) in simulation2 using topology 1

5.3.3.4 Simulation 2 with Topology 2

The queue length of node0 and node5 stay at high level due to the single MAC rate. The numbers of one hop neighbors are not the same as simulation2 with topology1. Still the results of the number of received packets per second are not correct. The values of node6 do not change with the sending rate of node5. We will discuss this in section 5.4.2

The Number of Packets Sent per Second

The results of simulation2 with topology2 are shown in Figure 5-10. The number of packets sent per second of node5 increases 100 packets every 10 seconds at the first 190 seconds, and then decreases 100 packets every 10 seconds from the 201th second to the 261th second. The number of packets sent per second of node1, node2 and node3 is changing continuously during the simulation.

The Number of Received Packets per Second

Because node1 can not communicate with node5 and node2 can not communicate with node6, there is less interference between node1, node5, node2 and node6. The results of the number of received packets per second are different from the results in section 5.3.3.3. The curve of node6 is smoother. And the number of received packets per second of node4 varies between 50 and 70 packets per second during the simulation.

The Queue Length

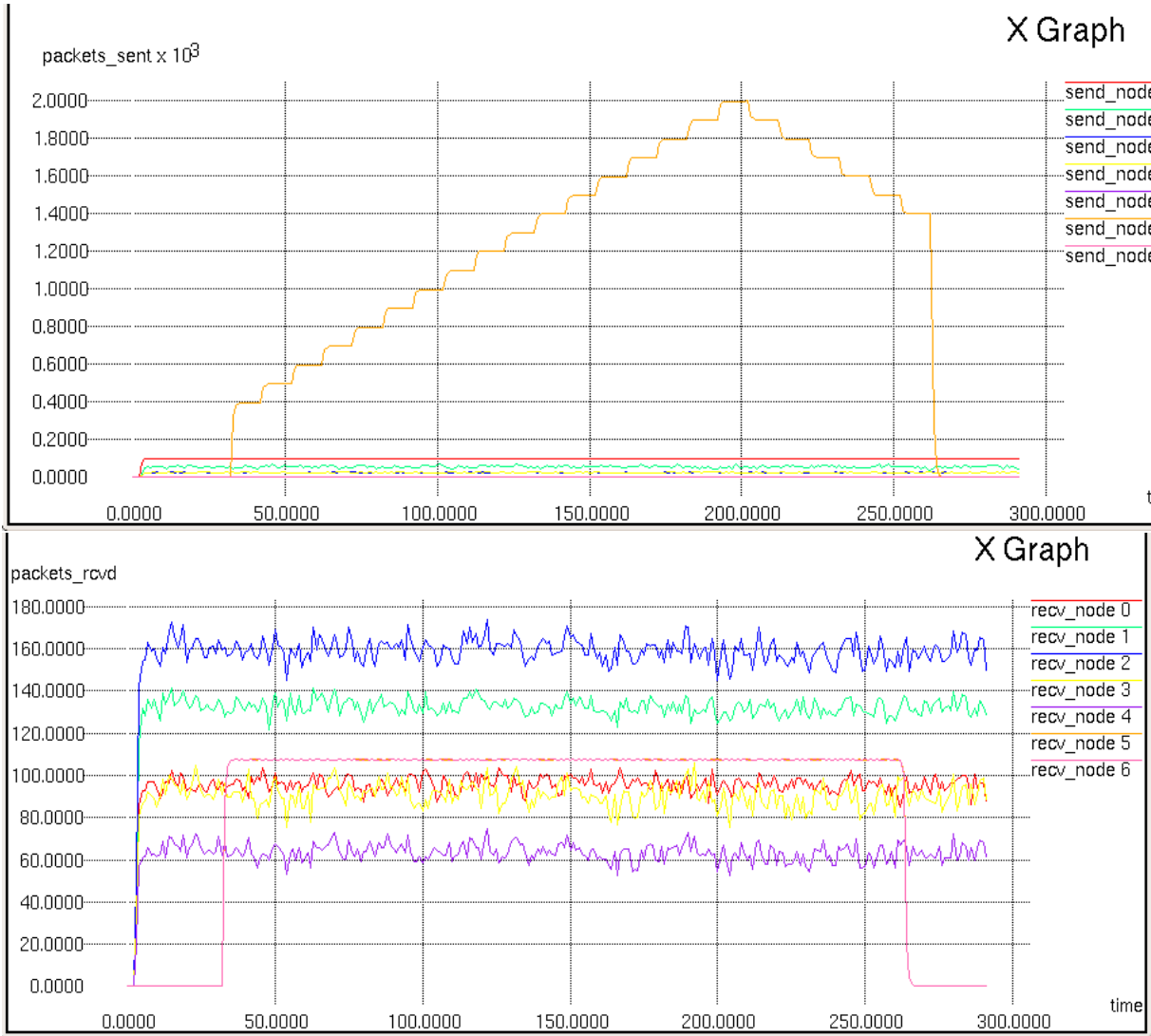
Due to the single MAC rate, the values of queue length of node0, node1 and node5 reach the highest level after the CBR traffics starts. Because there is less interference between node1 node5, node2 and node6, the link quality between node1 and node2 is much better than that in section 5.3.3.3. The curve of node0 is much smoother.

The Busy Time and the Number of Busy Period

The number of busy period of node6 rises quickly due to the rising of the packets sent per second of node5. And these values are much lager than the values in section 5.3.3.2. The number of busy period of node6 increases with the rising sending rate of node5 too. And the number of busy period of node6 is higher than others.

The Number of One Hop Neighbors

The change of the number of one hop neighbors is similar to section 5.3.3.2



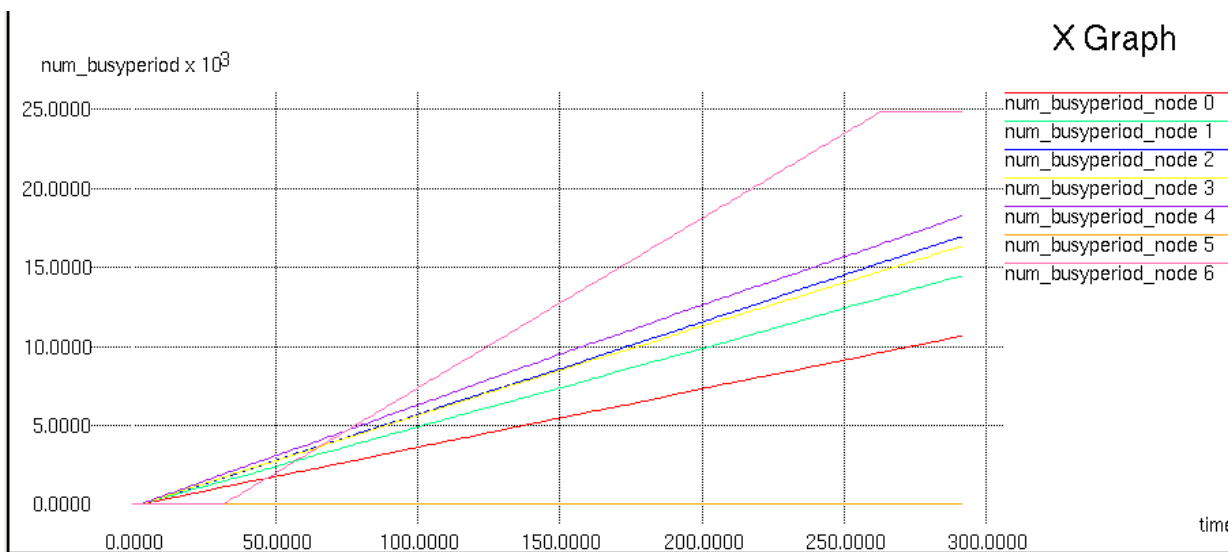
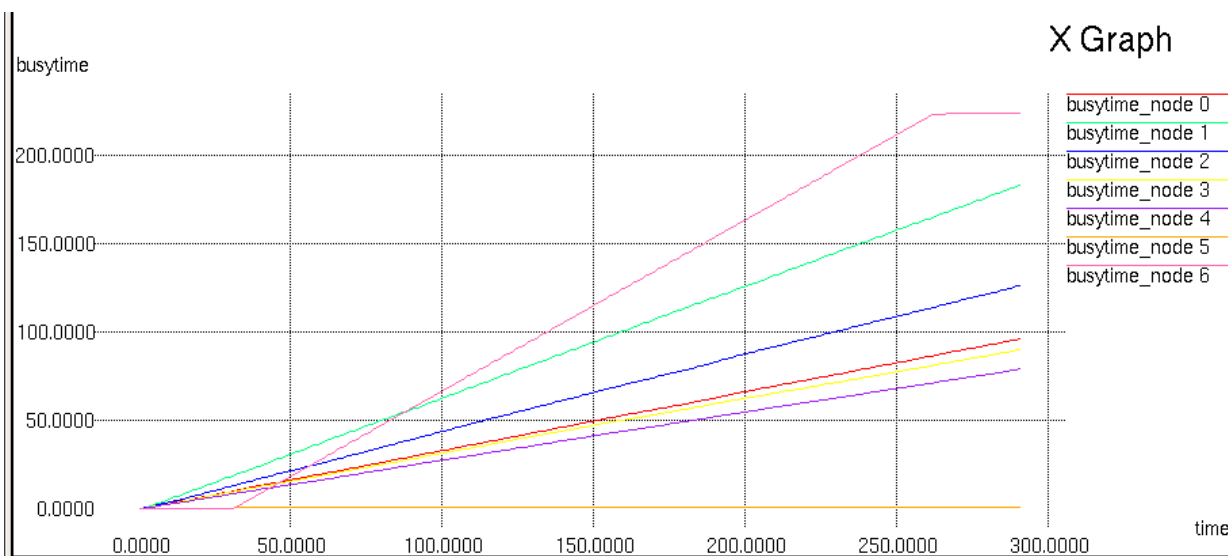
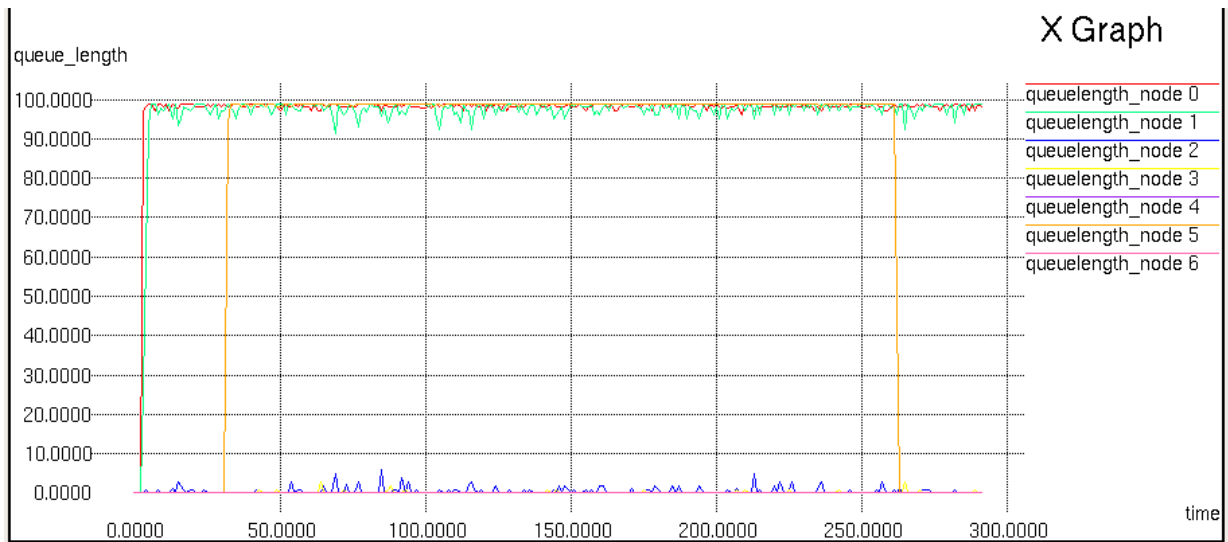




Figure 5-10 The Values of The Metrics (number of packets sent per second, number of packets received per second, busytime, number of one hop neighbors, number of busyperiod, queue length) in simulation2 using topology 2

5.4 Discussions

In section 5.3.3.3 and section 5.3.3.4, we note that the number of received packets per second of node6 does not change correspondingly. And the values always stay at a low level (108packets/second) during the run time. Even when node1 can not communicate with node5 and node2 can not communicate with node6, the throughput of CBR2 traffic still keeps low. Since both MAC/802.11/Multirate and MAC/802.11/Miracle use a same module to calculate the number of received packets, we assume the reason might be that MAC/802.11/Miracle always uses the base rate. We set up a scenario in NSMIRACLE to test this problem. The scenario has two nodes. The setting of the scenario is shown in Table5-2.

name	value
simulation time	61 second
PHY LAYER	
Carrier Sensing threshold	1.47E-011
Receiving power threshold	7.14E-011
Transmitter signal power	7.21E-003
Frequency	2.47E+009
Propagation Model	Freespace
Path loss	1
MAC LAYER	

RTS/CTS	off
Application1	CBR1
Packet size	1000Byte
Period 1 from 1 st second to 11 th second	0.05
Period 2 from 11 th second to 21 th second	0.02
Period 3 from 21 th second to 31 th second	0.01
Period 4 31 th second to 41 th second	0.005
Period 5 41 th second to 51 th second	0.002
Period 6 51 th second to 61 th second	0.001
distance between node0 and node1	160m
CBR1 start time	1rd second
CBR1 end time	61 th second

Table 5-2 Setting in a Test Scenario

The result is shown in Figure 5-11. From 1st second to 41th second, the number of received packets per second changes with the CBR rate. From 41th second to 61th second, the number of received packets per second varies between 106 packets/second and 108 packets/second. The result shows the maximum received packets per second using MAC/802.11/Miracle is 108 packets/second.

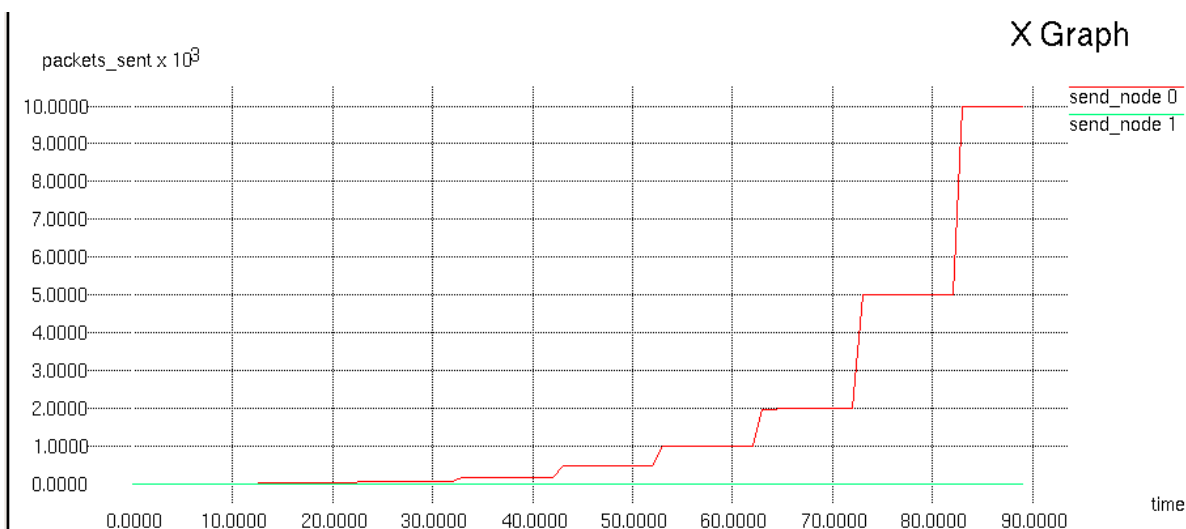


Figure 5-11 The Number of Packets sent per Second

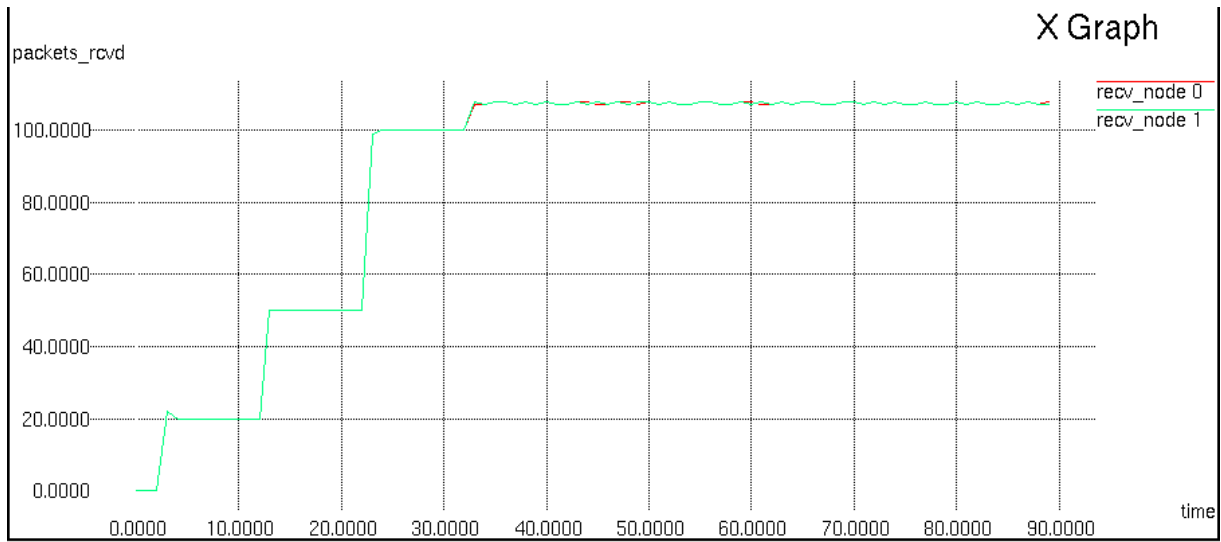


Figure 5-12 The Number of Packets Received per Second

6 Conclusion and Further Discussion

6.1 Conclusion

In this thesis, we proposed a distributed cross-layer monitor module based on NS2 with NSMIRACLE extension. This Cross-Layer Monitoring Module (CLMM) is the central instance for gathering and distributing the information. It has both time-driven and event-driven mechanisms for cross-layer monitoring. It has an action function to compare the values of metrics (i.e. queue length) with the thresholds, and trigger an action accordingly when the values are out of the thresholds. The modular structure of CLMM is flexible to define new events and their process functions that help to extend the event functions for different proposal.

We designed the CLMM based on NSMIRACLE framework in the project. We wrote new classes and modified the NSMIRACLE classes to realize the functions of CLMM. The two simulations show that the CLMM performs cross-layer monitoring as designed. It monitors metrics of the layers and queries the value of the metrics. It calculates the average value for the metrics and stores results to the storage module. When an event occurs it reacts by sending out a signal message and updating the metrics. Our simulations also show that these metrics of the layers interact with each other and one parameter of one layer affects other parameters of other layers during the communication.

By doing this project, we learned how to extent the NSMIRACLE framework to design a new module. We also learned how to use the new module for network performance analyst. We believed we archive the design goals as described in chapter 1. The project results a prototype of cross-layer monitoring module for NS2 with NSMIRACLE extension and it is ready to be used and extended for future work.

The project still has more work that needs to be done. Thus, we would like to propose some future work in section 6.2. The following section highlights some of ideas that can be extended to build a framework of the cross-layer monitoring module.

6.2 Future Work

6.2.1 Initialization Function for the CLMM

To establish simulations more easily, we propose the Initialization function that performs the initialization for the CLMM whenever it starts the monitoring. As shown in Figure 6-1, the Initialization Function has four sub-functions: Event_add, Event_handler, Metric_add and Message_add functions. The Event_Add, Metric_Add and Message_Add functions register events, metrics and messages for the CLMM. The Event_Handler function handles events dynamically. When an event is triggered, the Event_Handler function processes this event and updates its metrics.

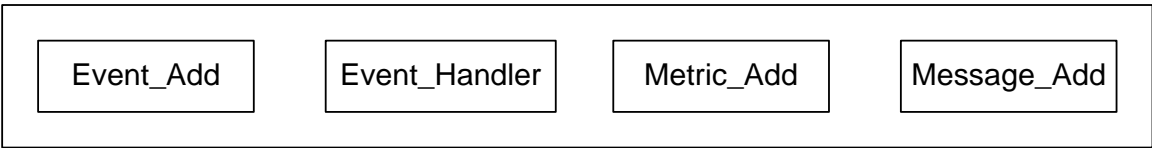


Figure 6-1 The Initialization Function

With the Initialization function, simulation is configured by a configuration file. The configuration file is not a module for the CLMM, but is used to set up CLMM environment. Before the CLMM starts monitoring, events, messages and metrics can be changed on demand. Thresholds also are added or modified according to different proposals. The configuration is loaded into the monitor module when the CLMM initializes the modules. Then the CLMM performs cross-layer monitoring based on the configuration. Using configuration file makes it easy to set up the CLMM environment without changing the main structure of the module. And it makes the module works in a more efficient way.

As shown in Figure 6-2, the configure file consists of three parts: the event list, the metric list and the message list. These lists include events, metrics and messages. When the CLMM starts initialization, the events, metrics and messages that are set up in the configuration file are loaded and transferred into event queue, metric queue and message queue correspondingly. The Initialization function registers these events, metrics and messages for CLMM.

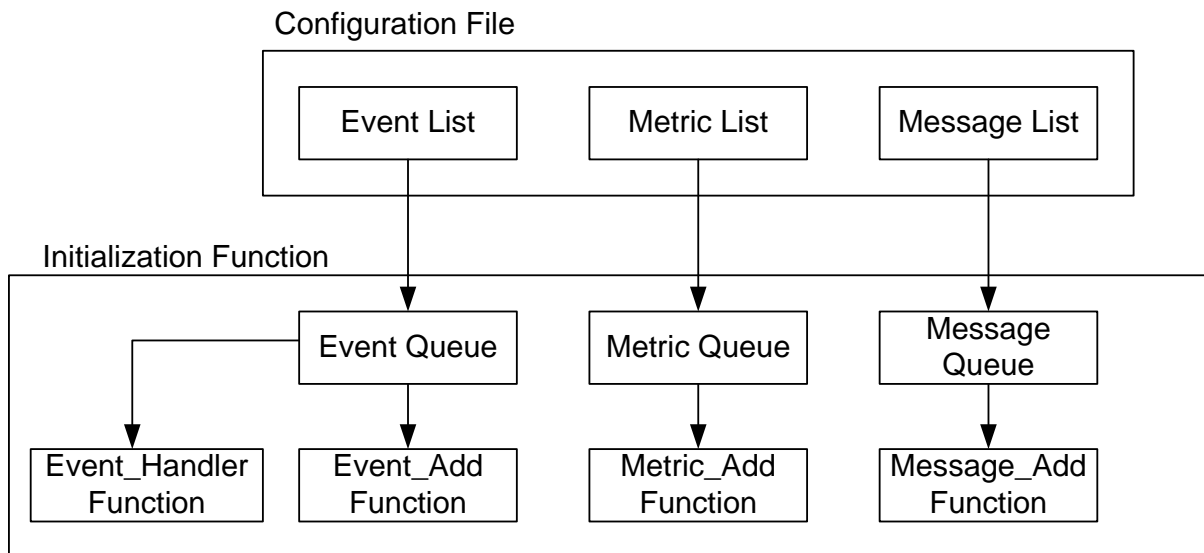


Figure 6-2 The structure of configuration file and the Initialization Function

6.2.2 The Error Function for the CLMM

Currently the error function only output an error message to the monitor module. However the errors function can be extended to two more sub-function: operation error control function and metric error function. The operation error control function monitors the status of the CLMM modules during the run time. When a status error occurs, the error function processes and responses it accordingly. The metrics error function contains error control functions for the metrics that handle all errors information of the layers or its neighbors. For example, if a queue length value is exceeded, the action sends an error message to the metrics error function, the error control operation is triggered when it receives the error message.

6.2.3 Action Module

The action module provides the action functions for the CLMM. The action functions are signaling and reaction functions which are based on event-driven or time-driven mechanism. The event-driven action function is similar to the event handler function implemented in layer-interface. However, instead of handling the events that come from the layers, the event-driven action function handles these events that come from its neighbor nodes. For example, when the uplink neighbor node receives a congestion signal message from its downlink node, it handles this signal message and raises a congestion event. Then action module can reduce its sending rate accordingly.

The action module can also take an action based on retrieving and comparing values of metrics. We set up the thresholds of some specific metrics (i.e. queue length or busy time) in

the action function. When the CLMM starts to monitor, the action function retrieves the values of the specific metrics from the storage function periodically, and then it compares the values with the thresholds. If the value is between the thresholds, the action function does nothing but keeps on retrieving and comparing. If the value is upper or lower than the thresholds, the action function triggers an action according to the configuration.

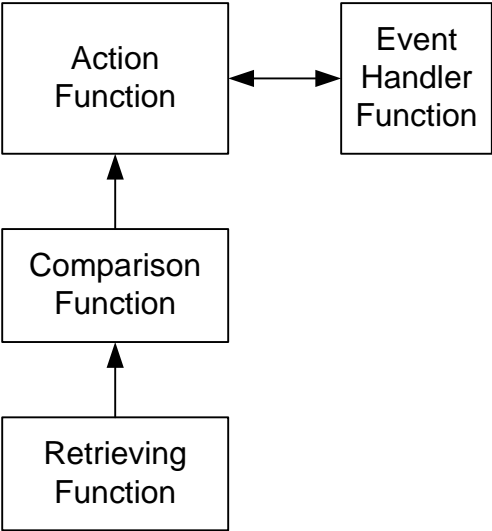


Figure 6-3 The Structure of the Action Module

As shown in Figure 6-3, the action module has four sub-functions: the action, retrieving, comparison and event handler function. The retrieving function retrieves the values of these metrics from the storage function periodically. When it gets the values, the comparison function compares these values with the thresholds. If it finds there is a value overloaded, it signals the action function to take an action. The event handler function has an event list. If an event occurs, first it checks if the event is in the list, if it is, it handles this event and signals the action function to take an action. The action function takes an action based on an event or a threshold.

With the action module, the cross-layer monitoring is extended to not only making the protocols collaborate, but also making the node's neighbors work together. It also makes it possible to observers how these nodes behaviors can interact with each other. Therefore we can optimize the network performance by adjusting parameters of the neighbor's nodes.

6.2.3 IP Routing Protocol

IP routing protocol not only provides routing metrics (such as the number of one hop neighbors) for the CLMM, but also supports the function that delivers a specific message to its neighbors. Delivering a message via a hello packet to its neighbors can make it possible that the neighbor's nodes know the status of the nodes. The message carrying the value of metrics or a signal is delivered periodically or triggered by an event-driven mechanism. The type of the message that is used to carry the values of the metrics for its neighbors is a routing message. Before being sent out, the routing message is packed into a hello packet. The node sends out the packet to its neighbors. With the message the node's neighbors have the detailed knowledge of the node and of the link crossing these two nodes.

The CLMM uses an IP routing agent to deliver a message to its neighbors. The IP routing agent is modified, therefore it can communicate with the message routing module. For the CLMM, the message sent to its neighbors is a routing message. When the message router receives a routing message, it simply transfers the message to IP routing agent. IP routing agent packs the routing message to a "Hello" message, and then it sends out the "Hello" message to its neighbors. The routing message from the source is marked with a specific ID, therefore each one hop neighbor of the nodes know where it comes from. When a node receives a routing message from its neighbor, first it checks if it has received this message already. If it has, the node drops the message. Otherwise the message is processed and sent to the monitor module. If the message is a signal message, for example, a congestion signal, and if the message is from downstream node, the monitor module reduces the data rate accordingly. The node keeps on monitoring the downstream node, if there is no congestion any more. The node increases the data rate to gain the maximum throughput.

With the routing module, the CLMM not only delivers a message to the layers, but also distributes messages to its neighbors. This makes it possible that each node not only shares metrics with the layers, but also shares the metrics with its neighbors.

The future work could be done by following steps:

- First, IP routing protocol is extended to process messages which go to or come from its neighbors.
- Second, the Action module is extended to make the protocols collaborate and the node's neighbors work together.
- The initialization function is added to make the configuration more easy and flexible.

6.2.4 Summary

With the above three extension modules, the CLMM could be extended to perform cross-layer monitoring based on the configuration, without changing the main structure of the module. And it could distribute the messages to its neighbor nodes, which makes it possible that each node not only shares metrics with the layers, but also shares the information of the metrics with its neighbors. Furthermore, based on the shared information, the action module could implement new reaction mechanisms to adjust the parameters for network performance optimization. We believe that the CLMM could be a useful tool for implementing and optimizing cross-layer mechanisms in NS2 with NSMIRACLE extension environment.

References

- [1]. I.F. Akyildiz, X. Wang, A Survey on Wireless Mesh Networks, *IEEE Communications Magazine* 43 (9) (2005) s23–s30.
- [2]. Ekram Hossain, Kin K. Leung, *Wireless Mesh Networks, Architectures and Protocols*, Springer, ISBN-10: 0387688382, ISBN-13: 978-0387688381.
- [3]. Vijay T. Raisinghani, Sridhar Iyer, *Cross Layer Design Optimization in Wireless Protocol Stacks*.
- [4]. Xudong Wang, Azman O Lim, *IEEE 802.11s Wireless Mesh Networks: Framework and Challenges*, *Ad Hoc Networks*, Vol. In Press.
- [5]. Joseph D. Camp, Edward W. Knightly, *The IEEE 802.11s Extended Service Set Mesh Networking Standard*, *IEEE Communications Magazine*, Vol. 46 Issue 8, p120-126, Aug 2008.
- [6]. G. R. Hiertz, S. Max, R. Zhao, D. Denteneer, and L. Berlemann, *Principles of IEEE 802.11s*, in *Proc. The First International Workshop on Wireless Mesh and Ad Hoc Networks (WiMAN 2007) in conjunction with 16th International Conference on Computer Communications and Networks ICCCN 2007*. Honolulu, HI, USA: IEEE Communications Society, Aug. 13–16, 2007, pp. 1002–1007.
- [7]. Guido R Hiertz, Sebastian Max, Yunpeng Zang, Thomas Junge, Dee Denteneer, *IEEE 802.11s MAC Fundamentals*, *Mobile Adhoc and Sensor Systems, 2007. MASS 2007. IEEE International Conference on (2007)*, pp. 1-8.
- [8]. *IEEE 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*
- [9] A. Jardosh, K. Ramachandran, K. Almeroth, and E. Belding-Royer, *Understanding Congestion in IEEE 802.11b Wireless Networks*, in *Proc. of IMC*, Berkeley, CA, Oct 2005.
- [10]. J. Li, Z. Li, and P. Mohapatra, *APHD: End-to-End Delay Assurance in 802.11e Based MANETs*, *MOBIQUITOUS 2006*.
- [11]. Srivastava, V.; Motani, M, *Cross-layer design: a survey and the road ahead*, *Communications Magazine*, IEEE Volume 43, Issue 12, Dec. 2005 Page(s): 112 - 119
Digital Object Identifier 10.1109/MCOM.2005.1561928
- [12]. Riggio R., Scalabrino N., Miorandi D., Chlamtac I., *JANUS: A Framework for Distributed Management of Wireless Mesh Networks*.

- [13]. Gupta, D.; Wu, D.; Chen, C.C.; Chen-Nee Chuah; Mohapatra, P.; Rungta, S. Experimental Study of Measurement-based Admission Control for Wireless Mesh Networks, Mobile Adhoc and Sensor Systems, 2007. MASS 2007. IEEE International Conference on Volume, Issue, 8-11 Oct. 2007 Page(s):1 - 9 Digital Object Identifier 10.1109/MOBHOC.2007.4428642
- [14]. H. Aiache, V. Conan, J. Leguay, M. Levy, XIAN, Cross-Layer Interface for Wireless Ad Hoc Networks. Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net), 2006.
- [15]. Ahn, G.S., Campbell, A.T., Veres, A., Sun, L.H., 2002. SWAN: Service Differentiation in Stateless Wireless Ad Hoc Networks. IEEE In Proc. INFOCOM'02, 1:457- 466.
- [16]. D. Gupta, P. Mohapatra, and C. N. Chuah, Efficient Monitoring in Wireless Mesh Networks: Overheads and Accuracy Trade-offs, IEEE Int. Conference on Mobile Ad-hoc and Sensor Systems (MASS), 2008.
- [17]. M. Petrova, P. Mähönen, J. Riihijärvi, M. Wellens, Cognitive Wireless Networks: Your Network Just Became a Teenager, in Poster Session of the IEEE International Conference on Computer Communications (INFOCOM), April 2006.
- [18] The GOLLUM-project website, <http://www.ist-gollum.org>, 2006.
- [19]. Nicola Baldo, Federico Maguolo, Marco Miozzoy, Michele Rossi and Michele Zorzi, NS-MIRACLE: a Modular Framework for Multi-Technology and Cross-Layer Support in Network Simulator 2.
- [20]. Michele Garetto, Jingpu Shi, and Edward W. Knightly, Modeling Media Access in Embedded Two-flow Topologies of Multi-hop Wireless Networks, In MobiCom '05, 2005.
- [21]. NS-2 http://nslam.isi.edu/nslam/index.php/Main_Page
- [22]. G. R. Hiertz, S. Max, T. Junge, L. Berlemann, D. Denteneer, S. Mangold, and B. Walke, Wireless Mesh Networks in the IEEE LMSC, in Proceedings of the Global Mobile Congress 2006, Beijing, China, Oct. 2006, p. 6.
- [23] NS-MIRACLE, URL: <http://www.dei.unipd.it/wdyn/index.php?IDsezione=2434>
- [24] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris, A High-Throughput Path Metric for Multi-Hop Wireless Routing, in ACM Mobicom, 2003.
- [25] Richard Draves, Jitendra Padhye, and Brian Zill, Routing in Multi-Radio, Multi-Hop Wireless Mesh Networks, in ACM Mobicom, 2004.
- [26]. Chung, J. and Claypool, M. NS by Example. <http://nile.wpi.edu/NS/>.
- [27]. Kevin Fall, Kannan Varadhan, The NS Manual, November 1, 2008.

[28]. Michele Garetto, Theodoros Salonidis, Edward W. Knightly, Modeling Per-flow Throughput and Capturing Starvation in CSMA Multi-hop Wireless Networks. IEEE/ACM Transactions on Networking, 16(4):864-877, August 2008.

[29].Erik Andersson, Emil Ljungdahl, Porting AODV-UU into NS-Miracle, <http://www.cs.kau.se/cs/prtp/pmwiki/pmwiki.php?n=MeshWikipage.AODV-UUForNs-miracle>.