A TCP-DRIVEN RESOURCE ALLOCATION SCHEME AT THE MAC LAYER OF A WIMAX NETWORK

by

Yu-shan (Susan) Chiu
B.A.Sc, Simon Fraser University 2005

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

In the
School of Engineering Science

SIMON FRASER UNIVERSITY

Fall 2008

APPROVAL

Name: **Yu-shan Chiu**

Degree: **Master of Applied Science**

Title of Thesis: **A TCP-Driven Resource Allocation Scheme at the MAC Layer of a WiMAX Network**

Examining Committee:

Chair: **Shawn Stapleton**
Professor of Engineering Science

_____

Steve Hardy
Senior Supervisor
Professor of Engineering Science

_____

Tejinder Randhawa
Supervisor
Adjunct Professor of Engineering Science

_____

Jie Liang
Internal Examiner
Assistant Professor of Engineering Science

Date Defended/Approved: _____

ABSTRACT

The paradigm of a traditional wired network protocol stack is a hierarchy of services provided by each layer, but its ability to handle an error-prone physical medium is severely compromised in wireless networks. Several approaches, including cross-layer techniques have been developed to address this problem. While much cross-layer research endeavour focused on interactions of the lower layers, in this thesis, I present a TCP to MAC cross-layer technique in a simulated WiMAX network. Using this cross-layer method, the scarce radio resource is intelligently distributed among stations, based on the information of congestion window size passing down from TCP. Both analytical and simulation models were developed to understand the behavioural dynamics of the proposed scheme, and quantify the performance gains. My results show that the proposed algorithm delivers a better performance in average end-to-end delay, file download time, and throughput when the traffic intensity of the network is moderate to high.

Keywords: Cross-layer; MAC scheduling; TCP; WiMAX; 802.16

Subject Terms: Wireless metropolitan area networks; IEEE 802.16 (Standards); Broadband communication systems

ACKNOWLEDGMENTS

I would like to thank Professor Hardy for the support that he has given to me throughout my Master study. His guidance, patience, and encouraging words helped me in every step of my path to the completion of this thesis. I would also like to thank Professor Randhawa for his innovative ideas, help and effort that he has put in even though he has other commitments. I would also like to thank Professor Liang for being the internal examiner of my thesis. I am grateful for my parents' faith in me, and their patience during my lengthy Undergraduate and Master study. To my lab mates, thank you all for your help and encouraging words. Last but not least, to my important friends, in Taiwan, US, UK and Canada, who have encouraged me. I sincerely thank your earnest 'add oil' words. Truly, when I needed it, they propelled me forward.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ACK	acknowledgment
AMC	adaptive modulation and coding
ARQ	automatic repeat request
ATM	asynchronous transfer mode
BE	best effort
BER	bit error rate
BS	base station
BWR	bandwidth request
CID	connection identifier
CSMA/CA	carrier sense multiple access with collision avoidance
cwnd	congestion window
DL	downlink
ertPS	extended real-time polling service
FFT	fast Fourier transform
FTP	File Transfer Protocol
HARQ	hybrid automatic repeat-request
IP	Internet Protocol
MAC	medium access control
MDRR	Modified Deficit Round Robin

MSS	maximum segment size
nrtPS	non-real-time polling service
OFDM	orthogonal frequency-division multiplexing
OFDMA	orthogonal frequency-division multiple access
OSI	Open System Interconnection
PDU	protocol data unit
PHY	physical (layer)
PMP	point-to-multipoint
QoS	quality of service
RTG	receive/ transmit transition gap
RTO	retransmission timeout
rtPS	real-time polling service
RTT	round-trip time
SACK	selective acknowledgment
SDU	service data units
SF	service flow
SFID	service flow identifier
SNR	signal-to-noise ratio
SS	subscriber station
ssthresh	slow start threshold
TCP	Transmission Control Protocol
TDD	time division duplex

ToS	type of service
TTG	transmit/receive transition gap
UGS	unsolicited grant services
UL	uplink
VoIP	voice over IP
WFQ	Weighted Fair Queuing
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	wireless local area networks
WMAN	wireless metropolitan area network

CHAPTER 1: INTRODUCTION

One of the features that differentiate wireless networks from wired networks is the scarce radio spectrum and time-varying channel quality. A great amount of research effort has been expended in attempting to improve the performance of wireless networks over what is an inherently error-prone medium for signal propagation. In a traditional network protocol stack, Transmission Control Protocol (TCP) often acts as the fundamental mechanism to ensure a reliable link between end-to-end stations. Unfortunately, while the architecture of TCP enables it to perform well over wired networks, it exhibits serious shortfalls when deployed over wireless networks.

Several research endeavours have attempted to tackle the challenges that arise when using TCP over wireless links. Sardar *et al.* [1] contributed a thorough and organized survey of various TCP enhancements for last-hop wireless networks. Cross-layer communication within the protocol stack was a concept that gradually emerged when researchers began to look beyond TCP or a single layer itself, and considered the possibilities of communicating among multiple layers to attain better performance outcomes.

Srivastava *et al.* [2] presented a definition and illustrated different kinds of cross-layer designs. A cross-layer design for the protocol stack permits direct communication and variable sharing between nonadjacent layers in an otherwise restricted protocol stack model. A cross-layer design could involve creation of

new interfaces, at which the new interface acts as an agent distributing information back and forth between the nonadjacent layers. However, the presence of an agent is not mandatory; it is possible to couple two or more layers during the design phase such that the functionalities of other layers are taken into account when the protocol operates. Another methodology of achieving cross-layer design is to merge adjacent layers into a *superlayer*. Finally, a cross-layer design can also be achieved by vertical calibration across multiple layers. Both [2] and [3] summarize a number of strategies to implement cross-layer communications in a protocol stack.

It is not the intent of this thesis to include all past cross-layer designs. Instead, I will outline a few in the following paragraphs. Due to the distinctly unreliable trait of wireless channels, numerous research efforts concentrating on achieving cross-layer capability have focused on the interaction between the physical (PHY) layer and higher layers. Thus, this outline will be organized in a lower layer to upper layer sequence.

Song *et al.* [4] proposed a dynamic subcarrier assignment scheme in an orthogonal frequency-division multiplexing (OFDM) wireless broadband network. Considering the advantages of multiple subcarriers in OFDM and multi-user diversity in a network cell, certain subcarriers may be in deep fade for one user but may not simultaneously be in deep fade for other users. The authors determined the available data rate of each subcarrier based on channel state information, and dynamically assigned the subcarriers to users according to a utility function to differentiate quality of services (QoS). The utility function in this

context represents the level of satisfaction of an end-user, which may differ depending on user-applications.

Toumpis *et al.* [5] presented a cross-layer power-control scheme for wireless ad hoc networks, which utilized the mechanism of the ad hoc medium access control (MAC) layer as the basis for determining the transmission power of a packet. Nodes that successfully capture the channel will transmit at a minimum sustainable power as specified by the control packets from the intended receiver, during the previous contention period. The energy consumed per packet is thus reduced, in comparison to the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) scheme. In addition, the reduction in transmission power diminishes the interference experienced by other nodes, thus increasing the probability of those nodes successfully capturing the channel. The overall result of this methodology is improvement in throughput at a reduced transmission power per packet.

Adaptive modulation and coding (AMC) is a technique that has been demonstrated to successfully deal with the time-varying transmission quality of the wireless channel. The modulation and coding scheme at the PHY layer adapts according to the channel state, in order to achieve a designated data rate within a certain bit error rate (BER) constraint. A number of cross-layer proposals were built on top of AMC, each of which incorporated an automatic repeat request (ARQ) mechanism at the MAC layer to improve the spectral efficiency in terms of bits per transmitted symbol [3]. Similar to AMC, the coding rate of a video streaming application can adapt depending on the channel condition and/or

the MAC layer performance to attain a QoS determined by an end user or the application.

Moving back to the upper layers of the network protocol stack, the problematic performance of TCP when deployed over wireless networks is a topic of research that attracts much attention. The explicit congestion notification (ECN) [6] enhancement for TCP captured the essence of the cross-layer concept. In ECN, a router incorporates active queue management to detect congestion before the queues of the router overflow. The router then explicitly indicates the congestion condition during the incipient congestion phase by marking the header field of a packet which originates from an ECN-capable network. TCP of the sender then initiates its congestion avoidance mechanism upon the detection of a marked packet. Consequently, TCP is capable of differentiating between an error over the wireless link due to degraded channel quality and one due to network congestion.

Kliazovich *et al.* [7] introduced a Snoop-alike agent, which was situated in between TCP and the MAC layer at both sender and receiver nodes. In the IEEE 802.11 Standard, the reliable delivery of data is established using the ARQ mechanism. The authors argued that in such scenario, the number of acknowledgments (ACKs) required for a single transmission is three-fold. The first ACK is the indication of successful delivery of the data packet at the MAC layer. The second ACK is rooted from the ACK generation of TCP, and the third ACK originates from the MAC layer to indicate the successful transmission of the TCP ACK. To eliminate duplicate confirmations of ACKs, their proposed agent

generates a TCP ACK locally at the sending node upon receiving the first indication of successful delivery of the data packet at the MAC layer. On the other hand, the agent at the receiving node intercepts the ACK generation by dropping the TCP ACK. The bandwidth usage is thus economized, but at the same time, the end-to-end semantic of TCP is violated.

Park *et al.* [8] introduced a notion of *channel access cost*, which was estimated based on the aggregated traffic load and per-station bandwidth usage evaluated at the MAC layer. A pseudo random number that is uniformly distributed between zero and one is generated and compared to the access cost. If the random value is less than the access cost, the *high access cost indicator* is set. An *upward notification* that discourages the participation of the station is conveyed to TCP to reduce its sending rate. In order to minimize the changes required to the original protocol stack, the notification is sent to the Internet Protocol (IP) layer. The ECN mechanism is then utilized to activate the congestion algorithm of TCP. On the other hand, if the access cost is low, the station is encouraged to participate more by elevating the sending rate of TCP, but this scenario was left open for future extension.

Yang *et al.* [32] proposed an asymmetric link adaptation for TCP-based applications in a WiMAX network. Due to the asymmetric load of uplink and downlink traffic of a TCP-based application, an aggressive modulation and coding scheme is utilized in combination with the ARQ mechanism of the MAC layer in the downlink to achieve an uncompromised TCP performance in a wireless network. In the uplink, a conservative modulation and coding scheme is

employed instead, to ensure the robustness and response time of the return ACK packets. In addition, a scheduler that attempts to maximize the benefit of the queue weight, data rate, delay, and queue size is dedicated to the best-effort service type of WiMAX.

Giambene *et al.* [9] contributed a diverse study on the use of cross-layer designs in satellite communications. In Section 9.4 of [9], the authors proposed a novel TCP-driven dynamic resource allocation scheme, where the resource allocated at the MAC layer is dependent on the condition of TCP. In particular, they formulated a prediction relating the growth of resource requests on a per frame basis to the change in congestion window size in TCP. The MAC layer then reallocates the resources based on this prediction before TCP reacts to this growth in congestion window size. This traffic prediction is useful in a high bandwidth-delay product network, such as a satellite network, due to high end-to-end response time. However, the same prediction process provides only limited advantages in a terrestrial network.

In this thesis, I have devoted my research effort to an alternative cross-layer scheme, inspired by the work of Giambene *et al*. TCP plays a key role in achieving the best possible level of a predefined end-to-end performance metric. Nevertheless, TCP operating independently cannot deliver optimum performance without being supplemented by knowledge from the layers below. In a wireless environment, lower layers are able to respond better to fast fluctuating channel states than TCP. To rectify this flaw of restricted sharing of information within the

protocol stack, I propose a scheme that allows TCP to communicate with lower layers, in particular the MAC layer.

In the proposed scheme, the MAC layer scheduler will adjust its service resources dedicated to each TCP flow according to the congestion window size received from TCP. In other words, the MAC scheduler adaptively allocates the service resources based on the network condition estimated by TCP. The subtle advantage of incorporating the MAC layer with TCP instead of the PHY layer is that TCP assesses the network condition at the end-to-end host level, whereas the PHY layer only evaluates the air-link quality at the last-hop.

The MAC layer technology employed in both [7] and [8] were wireless local area networks (WLAN). However, since the broadband networks are evolving to carry expanding numbers of multimedia applications, a cross-layer optimization specific to a terrestrial network is necessary. WiMAX is chosen as the wireless network platform for my study because WiMAX is projected to be one of the contenders for the next generation wireless metropolitan area network (WMAN). The OFDM and orthogonal frequency-division multiple access (OFDMA) property of WiMAX allows it to deliver high data rates with great flexibility. In addition, the WiMAX has a number of QoS parameters defined at the MAC layer, which allows its scheduler to deliver differentiated services to the clients. The proposed scheme is specific to TCP-based application but not limit to best-effort service type of WiMAX. This thesis employed OPNET® WiMAX model as the simulated model for the WiMAX network.

This thesis introduces background knowledge on the relevant layers (i.e. TCP and WiMAX) of the network protocol stack in Chapter 2, followed by a detailed algorithm and analysis on the proposed technique in Chapter 3. Chapter 4 outlines the implementations of the proposed technique in the OPNET models. The simulation results are presented and discussed in Chapter 5. Finally, the thesis concludes with a summary of contributions and future extensions in Chapter 6.

CHAPTER 2: RELEVANT LAYERS OF THE PROTOCOL STACK

The two protocol stack models that are most widely referenced in networking are the Open System Interconnection (OSI) model and the Internet protocol suite, which are commonly referred to as the 7-layer and the 5-layer model respectively. The OSI model is descended from the Internet protocol suite, inheriting its original architecture, while possessing additional functionality defined at the end-host application level. Since the focus of this thesis is on the cross-layer technique that transfers data across the network, I will be using the 5-layer Internet protocol suite as the reference model.

The Internet protocol suite consists of five layers from top to bottom: application, transport, network, data link and physical. The host layers reside on the end-hosts, which carry out commands issued by end-users, and provide data transport services at the end-to-end host level. In contrast, the media layers are mostly distributed in the core of networks, at which they conduct data transmission in the network core, and physically deliver data across links. Each layer is assigned a dedicated term as illustrated in Figure 2.1 [10] to represent its protocol data unit (PDU). This thesis will address the data units at each layer according to the Figure, and the term ‘packet’ will refer to a general formatted block of data in a packet-switch network.

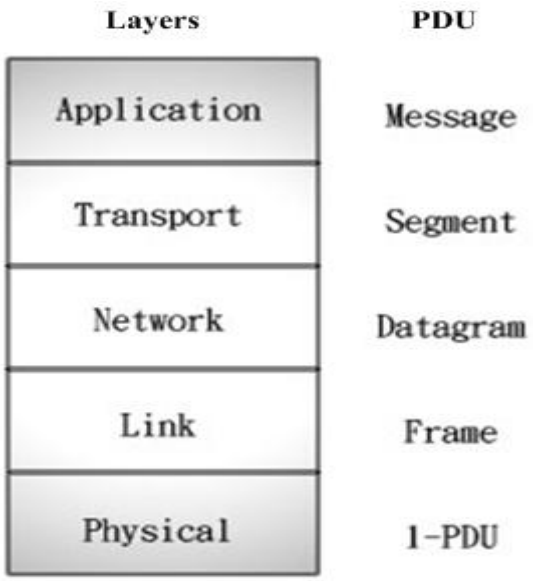

Figure 2.1: The Internet protocol suite

Since the proposed cross-layer technique involves modifications only at TCP and WiMAX, the rest of this chapter will be devoted to providing background information on these two layers. Most of the material discussed in the TCP section is based on W. R. Stevens' TCP/IP book [11] and J. Kurose's computer networking book [10].

2.1 TCP in a Nutshell

TCP is the most commonly utilized transport protocol in networks, and it provides a reliable and connection-oriented data transmission channel between end-to-end hosts. TCP establishes the reliable end-to-end transport of data by the use of sequence number and acknowledgment mechanisms. Messages passing down from the application layer are encapsulated into TCP segments, each of which is marked with a sequence number. The sequence number identifies the byte number of the first byte of each application data in the segment.

As multiple segments traverse across the network from sender to receiver, TCP at the receiver station identifies which segment is received based upon the sequence number. It then demultiplexes the segments in succession and passes the assembled message up to the application layer. The receiver in turn generates acknowledgments (ACKs) back to the sender upon the receiving of these segments. Consequently, the sender realizes whether a segment has been successfully transmitted to the destination and in sequence.

2.1.1 TCP Congestion Window

TCP employs a sliding window mechanism to control its sending rate. The size of the sliding window corresponds to the maximum amount of data that can be sent into the network before being acknowledged. The window slides across a stream of sequenced data in an ascending order as acknowledgments are returned to the sender. Upon receiving these acknowledgments, the window opens in size to increase its sending rate. Figure 2.2 illustrates the concept of sliding windows in TCP.

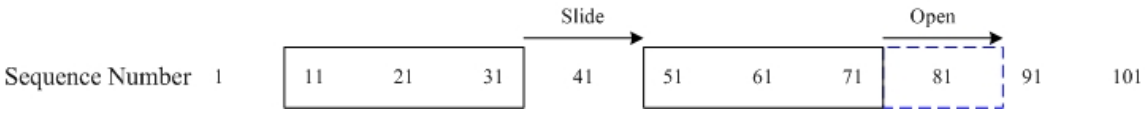

Figure 2.2: The sliding window

In TCP, the parameter representing the size of the sliding window is denoted as the congestion window (*cwnd*), and this parameter is possessed and controlled by TCP on the sending node. The unit of *cwnd* is maintained in bytes and is initialized to one maximum segment size (MSS) at the beginning of a

transmission session. The MSS is the largest size of a segment in bytes that TCP will send, and this information is exchanged between sender and receiver at the connection-establishment time. The growth of *cwnd* is dependent on the number of acknowledgments received, and is divided into two phases: slow start and congestion avoidance.

2.1.2 TCP Slow Start

TCP begins a data transmission session with the slow start phase. In slow start, TCP initially assumes that the network is congestion-free, thus aggressively increasing *cwnd* exponentially. This is done in the hope achieving the optimum performance faster. The growth of *cwnd* in slow start phase is described in Equation 2.1, where *segsize* represents the size of one segment in bytes.

$$\Delta cwnd = segsize \times number\ of\ \ ACKs\ returned \qquad (2.1)$$

Segsize is used instead of MSS to incorporate the possibility of encountering a smaller maximum transmission unit (MTU) along the transmission path between sender and receiver. This path MTU could limit the actual segment size to a smaller value than the MSS that was initially announced by the two end-hosts.

Consider the possibility that packets are never lost in a network, and the *cwnd* is initialized to one *segsize* at the beginning of a session. *Cwnd* is incremented by one *segsize* as the first segment is successfully transmitted and acknowledged, resulting in a growth from one to two *segsizes*. In the second round, two segments are sent and acknowledged resulting in two *segsize*

increments in addition to the original 2 *segsizes*. The process continues, and consequently results in an exponential rise in *cwnd* in every round. Despite an aggressive injection of data during the slow start phase, TCP still takes precautions to avoid flooding the network. To avoid this possibility, TCP enters the congestion avoidance phase, and increments *cwnd* in a more conservative fashion.

2.1.3 TCP Congestion Avoidance

The slow start and the congestion avoidance phases are differentiated by the result of a comparison of the *cwnd* value to a slow start threshold value, *ssthresh*. TCP operates in the slow start phase if the value of *cwnd* is smaller or equal to *ssthresh*; it operates in the congestion avoidance phase if otherwise. The change of *cwnd* in the congestion avoidance phase is described in Equation 2.2.

$$\Delta cwnd = \min\left(\frac{segsize \cdot segsize}{cwnd} \times number\ of\ \ ACKs\ returned,\ \ 1\ segsize \right) \qquad (2.2)$$

The equation restrains the maximum increment in one round to one *segsize*, resulting in an approximately linear growth in *cwnd*. Figure 2.3 illustrates the fundamental ideas of TCP slow start and congestion avoidance phases. One round usually corresponds to one round-trip time (RTT), which is measured from the time that a segment leaves the sender node to the time an ACK that covers the segment arrives at the sender.

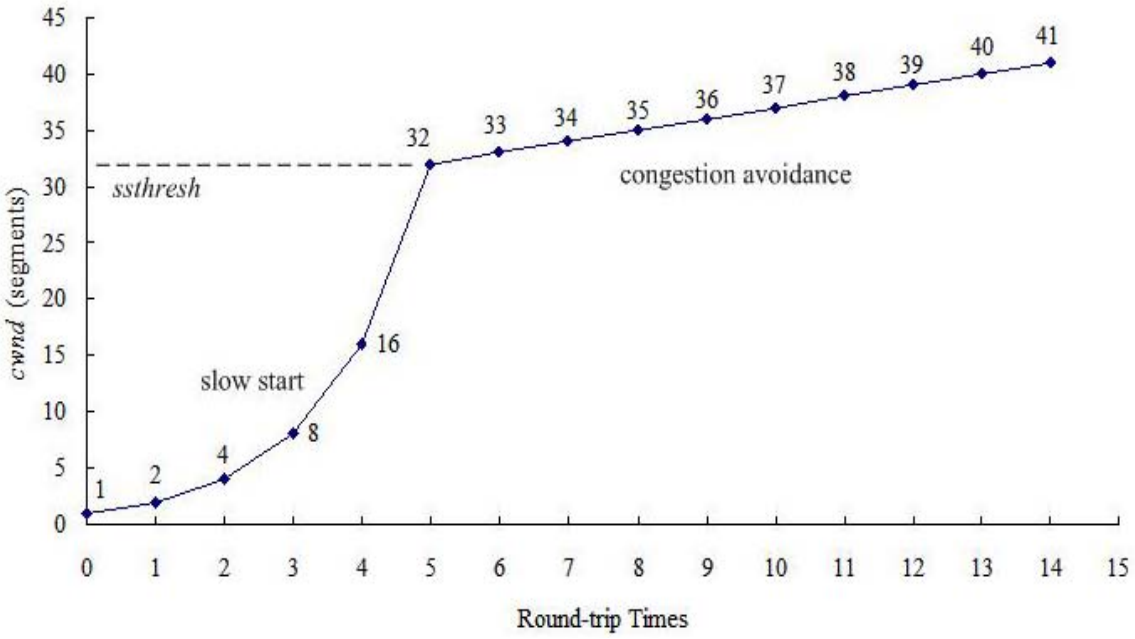

Figure 2.3: An illustration of the *cwnd* growth in TCP slow start and congestion avoidance phases

2.1.4 The Advertised Window

While *cwnd* continues to grow if no packet drop occurs, TCP on the sender node can never transmit more segments than the value of a window size advertised by the receiver. The advertised window indicated by the receiver represents the maximum buffer space that the receiver station is capable of sustaining. In the case of a fast server, fast transmission link and a slow client, this mechanism ensures that the server does not overwhelm the client with intense traffic, thus avoiding packet loss due to receiver buffer overflow. As a result, if *cwnd* is the flow control imposed by the sender, the advertised window can be viewed as the flow control imposed by the receiver.

2.1.5 Duplicate ACKs – TCP Fast Retransmit

A TCP segment includes an ACK number field to indicate the last consecutive segment of a data stream that was successfully received. In the event that one or more segments are missing from a window of transmitted packets, arriving segments which are sequenced after the first missing segment cause TCP on the receiver node to generate ACKs with the same ACK number as the previous one. These ACKs are referred to as *duplicate ACKs*.

Duplicate ACKs occur when segments are lost or arrive out of order; however, the number of duplicate ACKs generated due to an out of order arrival is significantly less than that of a packet loss event. Therefore, TCP utilizes the number of duplicate ACKs to infer a packet loss in the network. More specifically, when TCP detects three duplicate ACKs, it immediately retransmits the missing segment indicated by the duplicate ACKs, in an effort to recover a packet that is presumably dropped in the network. This mechanism is known as *fast retransmit*.

In the absence of fast retransmit, TCP receives duplicate ACKs but waits for a retransmission timer to timeout before resending the missing packet. This significantly reduces the performance of TCP because the range of a retransmission timer is typically in seconds as opposed to milliseconds for the detection of duplicate ACKs.

2.1.6 TCP Fast Recovery

Upon the activation of fast retransmit, TCP initiates the *fast recovery* algorithm. In fast recovery, *ssthresh* is set to one-half of the minimum between the current *cwnd* and advertised window values. The lost segment inferred from

the duplicate ACKs is retransmitted, and the new *cwnd* value is reset to *ssthresh* plus 3 times the segment size. Meanwhile, each *additional* duplicate ACK increments the *cwnd* value by one segment size. The motivation of this strategy is to encourage TCP on the sender node to continue sending data, thus maintaining the data transmission even during a packet loss event. As the ACK of the retransmitted segment is returned to the sender, it acknowledges the last in-order segment that has successfully arrived at the receiver, but queued in the buffer. After that, the *cwnd* is restored back to *ssthresh* plus 3 times the segment size. As a result, TCP enters congestion avoidance phase after fast retransmit instead of slow start, hence, the name *fast recovery*. Figure 2.4 is a figure from [11] with additional annotations to illustrate the changes in the values of *cwnd* and the sequence number of the segments sent during the fast retransmit and fast recovery phases.

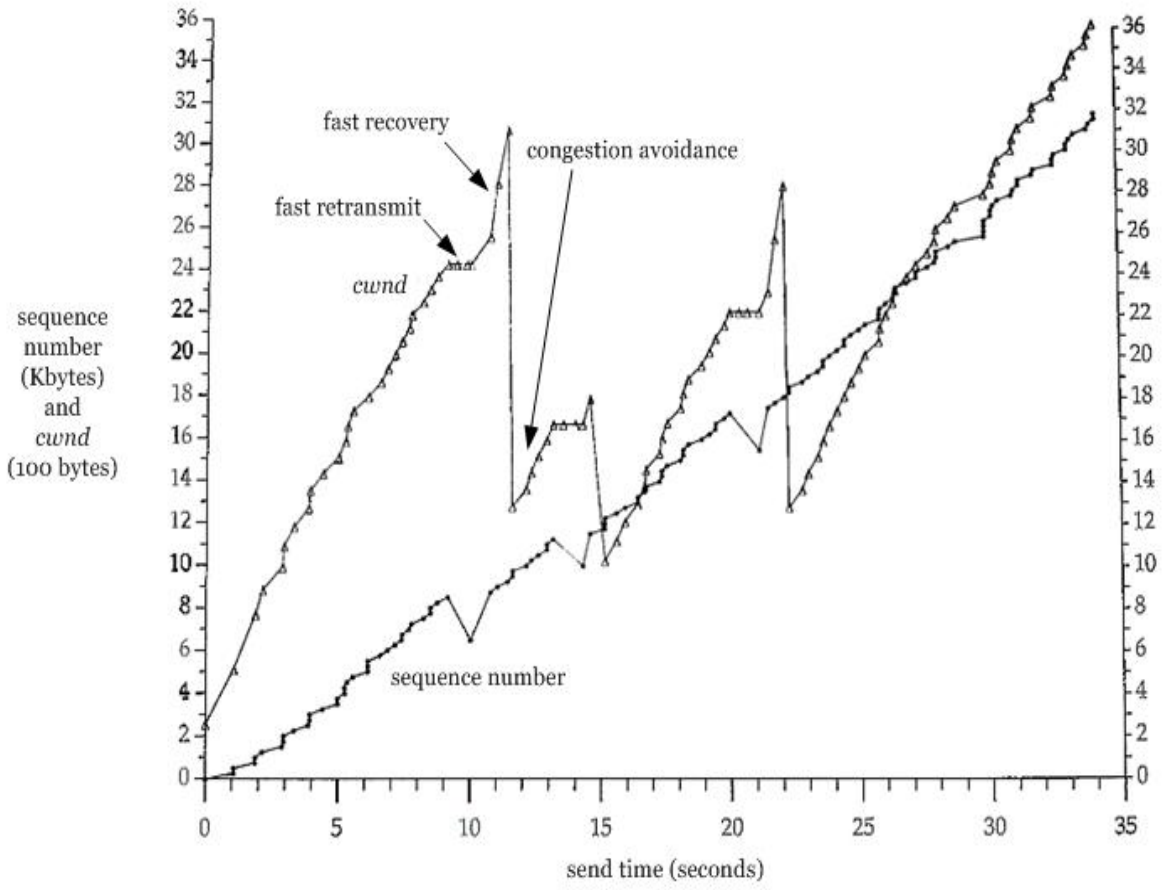

Figure 2.4: An illustration on the behaviour of *cwnd* and sequence number during the fast retransmit and fast recovery phases

2.1.7 TCP Timeout

Both fast retransmit and fast recovery are utilized to mitigate the effect of a packet loss event while TCP is in operation. However, fast retransmit and fast recovery can only be activated by a sufficient number of duplicate ACKs, in this case three. When a considerable amount of data in a window is lost, or when ACKs fail to return, a condition of insufficient duplicate ACKs received arises, leaving TCP and data transmission to idle. Under these circumstances, TCP relies on a timer to determine when to retransmit the missing packets, and when

to reactivate the transmission process. This timer is referred to as the retransmission timeout (RTO).

RTO is a parameter that is critical to TCP performance. A large RTO value leaves TCP idling longer than desired before retransmission occurs, thus prolonging the response time of TCP when encountering a loss event. On the other hand, a small RTO value gives rise to unnecessary retransmissions. TCP deems timeout to be a more serious consequence of network congestion than duplicate ACKs. Therefore, TCP throttles the traffic flow by reducing *cwnd* to one segment size each time after a retransmission timeout. Small RTO values result in unnecessary stalls in data transfer, thus degrading the performance of TCP.

The value of RTO is calculated based on estimations of RTT. For more details on RTT estimation and RTO calculation, please refer to Chapter 21 of [11].

2.1.8 Flavours of TCP

Section 2.1 has so far dealt with the fundamental operations of TCP; however, over the course of its development, TCP has been modified many times in attempts to improve the responses towards incidents of segment drops. These modifications are often referred to as different *flavours* of TCP. In this sub-section, I will outline a few selected flavours of TCP.

2.1.8.1 TCP Tahoe

TCP Tahoe was the first version of TCP to incorporate the slow start, congestion avoidance, and fast retransmit mechanisms. Nevertheless, the lack of the fast recovery mechanism caused Tahoe to enter the slow start phase every

time a packet is lost in the network, in spite of the fact that a loss is inferred by either a timeout or triple duplicate ACKs. As a result, Tahoe is not ideal when packet loss is significant.

2.1.8.2 TCP Reno

In addition to the mechanisms in TCP Tahoe, fast recovery was first introduced in TCP Reno in an attempt to assess different degrees of congestion levels in networks. It differentiates between a triple duplicate ACKs and a timeout event. As previously described in Section 2.1.6, Reno retransmits the lost segment upon activation of fast retransmit, and *cwnd* is incremented by one segment size for each additional duplicate ACK received. In Reno, the increases in *cwnd* lead TCP to send *new* segments during the fast recovery phase. In the event of a single packet drop in a window of transmitted data, Reno can quickly recover the dropped segment and keep data flowing. It thus performs better than Tahoe. However, Reno's major shortcoming is exposed when it encounters more than one segment drop in a window of transmitted data.

2.1.8.3 TCP New Reno

New Reno includes all mechanisms in TCP Reno, but the subtle difference between Reno and New Reno is the interpretation of *additional* duplicate ACKs received by the sender node after entering the fast recovery phase. New Reno assumes that the occurrences of packet drops are correlated. In other words, packet drops can happen consecutively, which is usually a legitimate assumption in wireless networks. Instead of proceeding with the

transmission of new segments upon receiving of additional duplicate ACKs in fast recovery, New Reno *retransmits* the segment immediately following the previously transmitted segment that has not yet been acknowledged. This interpretation becomes particularly useful when consecutive packets are dropped in a window of transmitted data. As a result, New Reno performs better than Reno in an environment that exhibits correlated packet drops such as wireless networks.

2.1.8.4 TCP SACK

The selective acknowledgment (SACK) outlined here is based on RFC 2018 [12]. In SACK, the received data is treated as blocks of data demarcated by missing segments. In other words, the TCP on the receiver node explicitly indicates which blocks of data it has received in the SACK option field of a TCP segment header. A block of data is defined by two edges, left and right, where a left edge corresponds to the sequence number of the first segment in a block. The right edge of a block is represented by the sequence number of the first missing segment immediately following the last received segment of the block. Provided with this information, TCP on the sender node is capable of retransmitting only segments that have not yet successfully arrived at the receiver, instead of wasting resources on retransmitting segments that are queued in the receiver's buffer. The SACK option is designed to remedy the flaw of TCP Reno in the wireless transmission environment, and it avoids unnecessary retransmissions, as it may be the case for TCP New Reno.

This ends the discussion of TCP in this thesis. The next layer that is involved in this thesis is known as the WiMAX MAC layer. Materials presented in the next section are based on three IEEE overview papers [13] [14] [15], the IEEE 802.16 standard [17] [18], and the OPNET documentation [16] [19].

2.2 WiMAX in a Nutshell

WiMAX is an acronym for Worldwide Interoperability for Microwave Access; it specifies a high bandwidth broadband technology for a WMAN. WiMAX is rooted from the IEEE 802.16 standard and is maintained by a non-profit industrial consortium called the WiMAX Forum®. The major task of the WiMAX Forum is to develop *WiMAX system profiles* that are complementary to the IEEE 802.16 standard, and to ensure that the devices developed based on these profiles are interoperable across manufacturers. Due to this close relationship, WiMAX and 802.16 are often referred to interchangeably.

The IEEE 802.16 standard was first drafted in 2001, and was initially intended to provide high bandwidth communication with line-of-sight for fixed wireless networks, operating at the 10-66 GHz frequency bands. Nevertheless, the amendment project that aimed to provide non-line-of-sight wireless communication, operating at the 2-11 GHz range, received much attention, and led to the completion of 802.16a-2003. The standard then introduced some enhancement features in the uplink, and evolved to 802.16-2004 (also known as 802.16d) that specified the technical details of the air interface and the access scheme of a fixed broadband wireless service. The mobility feature was later

added in the 802.16e-2005 amendment, which included significant antenna technology enhancements.

WiMAX, too, initially only specified profiles for fixed broadband wireless services, but it also underwent reviews to address the mobility issue. Hence, the mobile WiMAX was developed under the 802.16e-2005 specifications to support full mobility services. This thesis involves only modifications on the WiMAX MAC layer; however, since WiMAX is highly anticipated as one of the next generation wireless technologies, I will also outline a few key features of the WiMAX PHY layer.

2.2.1 The WiMAX PHY Layer

The IEEE 802.16 standard defines five specifications for the PHY layer, including WirelessMAN-SC, WirelessMAN-SCa, WirelessMAN-OFDM, and WirelessMAN-OFDMA. The first two specifications are defined based on the single-carrier technology, whereas the later two are defined based on the OFDM multi-carrier modulation. Over the course of the wireless development, OFDM is deemed as a more robust and efficient technology than the legacy single-carrier techniques for wireless networks. Therefore, WiMAX adopts both WirelessMAN-OFDM and WirelessMAN-OFDMA designs from the IEEE 802.16 standard, which are targeted for non-line-of-sight services operating at the frequency bands below 11 GHz. The flexibility offered by the OFDMA scheme makes it one of the most appealing features associated with WiMAX.

2.2.1.1 The OFDMA Technology

OFDMA is a multiple access scheme that builds on top of OFDM. Signals in OFDMA are transmitted with multi-carriers, in which each user is allocated with a subset of subcarriers for signal transmission and multiple access. The property of multiple subcarriers provides the OFDMA technology an inherent flexibility for sub-channelization across sectors in a network cell, and QoS differentiation among users.

One subchannel consists of a subset of subcarriers, of which the subcarriers in a subchannel can be distributed across the whole spectrum, or adjacently allocated as blocks of subcarriers. When subcarriers are adjacently allocated, it allows the use of AMC to achieve more efficient signal transmission. The QoS differentiation can be established by assigning a distinct code spreading factor to each subchannel, thus resulting in different transmission rates in each subchannel. Furthermore, the possibilities of dynamic subcarrier assignment and power adaptation depending on the subchannel condition and the QoS provisions can also be realized in an OFDM-based network.

OFDM is a frequency-division multiplexing technique that allocates subcarriers that are orthogonal to each other in the frequency spectrum. This reduces the inter-channel interferences, and allows the subcarriers to be closely spaced. The OFDM technology utilizes multiple low data-rate narrowband subcarriers instead of single rapidly modulated wideband carrier. The low data rate lengthens the symbol time, thus reducing the inter-symbol interference, and consequently leads to a simpler and more affordable equalization process. These

fundamental properties of OFDM make it robust, spectral efficient, and thus appealing to be deployed in a severe channel condition such as the wireless medium. Nevertheless, OFDM has a shortcoming due to its vulnerability towards Doppler shift effect because of the closely spaced subcarriers. This effect may become more serious as the mobility support is introduced to the standard.

2.2.1.2 Scalable OFDMA – Dynamic Channel Bandwidth

Scalable OFDMA is a form of OFDMA that has adjustable channel bandwidth based on the fast Fourier transform (FFT) sizes. The channel bandwidth changes while the actual subcarrier spacing remains fixed; only the grouping of subcarriers is changed. Table 2-1 shows the correspondences of the FFT sizes to channel bandwidth defined in the IEEE 802.16 standard.

Table 2-1: FFT sizes and the corresponding channel bandwidth in WiMAX

FTT Size	Channel Bandwidth
128	1.25 MHz
512	5 MHz
1024	10 MHz
2048	20 MHz

2.2.1.3 OFDMA TDD Frame Structure

When implementing a time division duplex (TDD) system in WiMAX, an OFDMA frame is divided into the downlink (DL) transmission period followed by the uplink (UL) transmission period. A preamble announces the initiation of a frame, followed by a transmit/receive transition gap (TTG) in between the DL and UL transmission periods, and finally a receive/transmit transition gap (RTG) in between two frames. During the DL transmission period, a *downlink map* (DL-MAP) is placed after the preamble to indicate allocations and burst profiles of the data bursts in the DL subframe. Similarly, an uplink map (UL-MAP), if available, contains the entire access information for the uplink. The frame structure can be viewed as two-dimensional space that spans across time (i.e. symbol time) in the x-axis and across frequency (i.e. subchannel or subcarrier) in the y-axis. Figure 2.5 [18] illustrates the WiMAX OFDMA TDD frame structure.

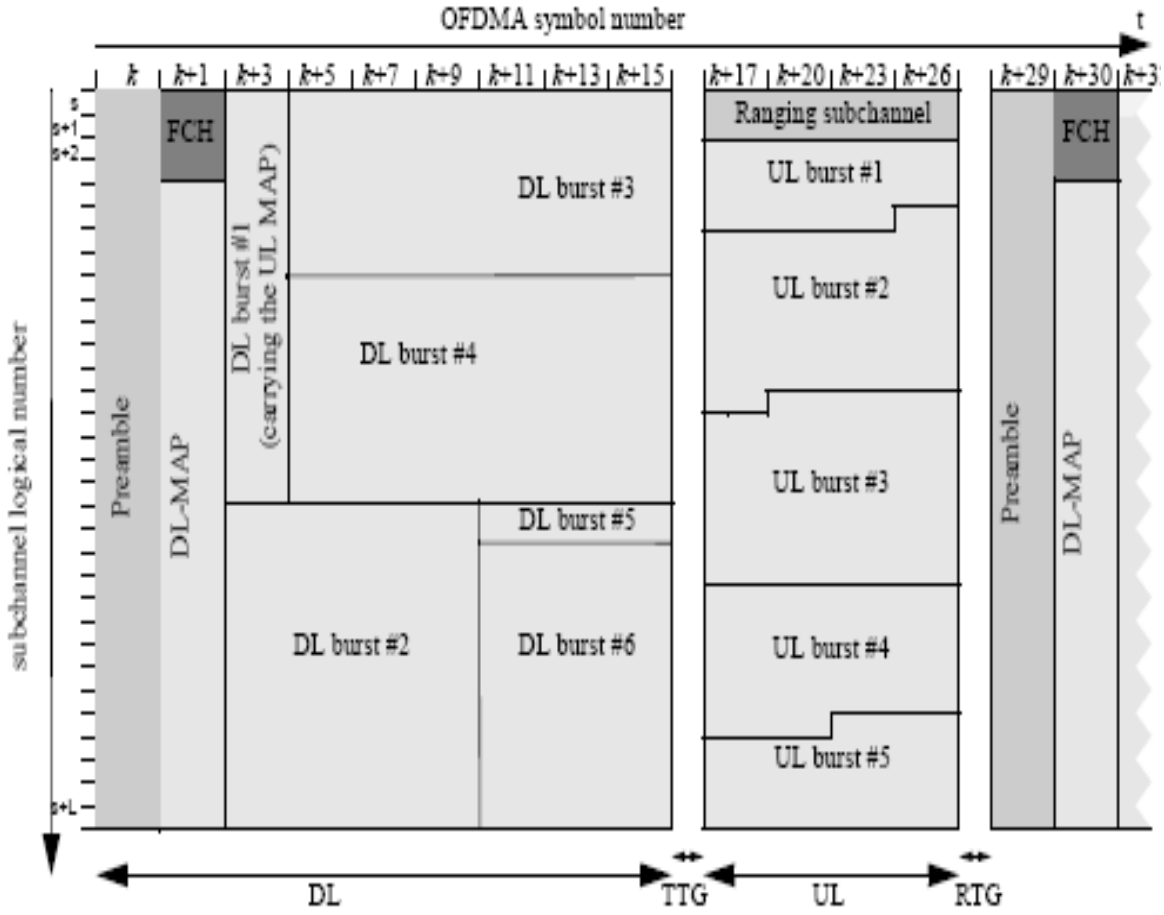

Figure 2.5: The WiMAX OFDMA TDD frame structure

2.2.1.4 Antenna Technology Options

Beside the intrinsic flexibilities offered by the OFDMA technology, WiMAX employs many advanced antenna technologies as options to enhance its PHY layer capability. They include multiple-input and multiple-output (MIMO) [25] and adaptive antenna system (AAS) [26]-[29] technologies. MIMO is a technique that uses multiple antennas at both the sender and receiver side to achieve multiplicative increases in data throughput without extra bandwidth or transmit power consumption. AAS is a system that utilizes multiple antennas, and

combines the antenna pattern and signal processing to reduce interference, thus improving the system capacity.

This sub-section has summarized a few key features in the WiMAX PHY layer. The next sub-section will describe the specifications in the WiMAX MAC layer.

2.2.2 The WiMAX MAC Layer

The WiMAX MAC layer is divided into three sublayers from top to bottom, the service specific convergence sublayer, MAC common part sublayer and the security sublayer. The convergence sublayer supports two types of services; one is for asynchronous transfer mode (ATM), and the other one is packet service for packet-switched networks. The common part sublayer provides utilities that are common to both types of services in the convergence sublayer. The major functionalities of the common part sublayer include but are not limited to network entry, connection management, QoS control, air-link control, PDU operation, mobility and power management, and multicast and broadcast services. This thesis does not intend to describe the MAC common part sublayer in whole, but will outline a few key features in the following sub-sections.

The security sublayer is the third sublayer in the MAC layer, which is immediate above the PHY layer. It is responsible for privacy, authentication and confidentiality for the subscriber stations in the network. The security sublayer is not relevant to the discussion of this thesis, thus will not be discussed further in this thesis.

2.2.2.1 Service Specific Convergence Sublayer

The major task of the convergence sublayer is to transform each higher layer PDU into a MAC service data unit (SDU), and map it to the appropriate *transport connection* according to a set of classification rules. Some examples of the classification rules includes the matching of IP source/destination addresses, application source/destination port numbers, and IP type of service (ToS) specifications. A matched packet is sent to a transport connection, which is referenced by a *connection identifier* (CID). A CID identifies a unidirectional transport connection between a base station (BS) and a *subscriber station* (SS). In other words, the CIDs of the DL and UL transport connections between the same BS and SS pair are unique. Figure 2.6 [17] illustrates the idea of packet classification in the convergence sublayer. Please note that the classification can be performed by either the BS or SS, depending on the transmission direction.

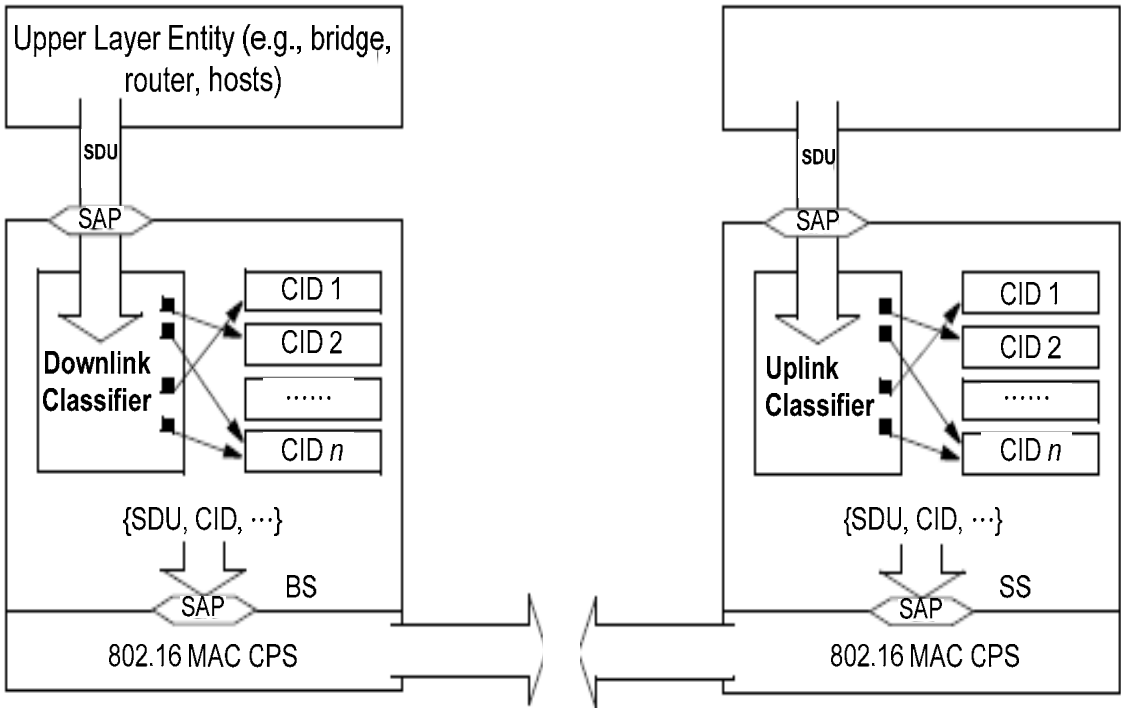

Figure 2.6: Packet classification of the service specific convergence sublayer

2.2.2.2 MAC Common Part Sublayer

The IEEE 802.16-2004 specifies two modes of operation in the MAC common part sublayer; one of which is the point-to-multipoint (PMP) mode, and the other one is the mesh mode. The PMP mode operates like a typical centralized network system, where a number of client stations (i.e. SSs) are connected to and served by a centralized server station (i.e. BS). The downlink transmission is broadcast in the network, whereas the uplink transmission is admitted on a demand basis. The frame structure is partitioned into DL and UL subframes as it is illustrated in Figure 2.5 in Section 2.2.1.3.

The mesh operation mode is organized in a similar fashion as an ad hoc network, in which each station is allowed to establish direct connection with each other. The frame structure has no explicit DL and UL subframe separation as in

the PMP mode. Nevertheless, the mesh mode is not the focus of this thesis, thus the following content will assume the mode of operation is PMP.

2.2.2.2.1 Network Entry

Upon the entry of a client into the network, the SS scans through its frequency list attempting to synchronize with a BS. The BS performs the admission control algorithm to decide whether to admit the SS, based on the QoS requirements requested by the SS and the current resource availability of the BS. If admitted, the BS generates a set of CIDs and connections, including the management and transport connections, to associate with the SS. Three pairs (i.e. DL and UL) of management connections are established for control packets, and the transport connection is used for data transmission. Along with the new CIDs, the BS also assigns new *service flow identifiers* (SFIDs) to the new data flows associated with the station. A *service flow* (SF) is a unidirectional flow of MAC SDUs between a pair of BS and SS, of which is provided with a specific set of QoS parameters, and an SFID uniquely identifies the service flow.

2.2.2.2.2 QoS Provision (802.16 standard: 6.3.5.2)

If OFDMA is deemed as the most important property of the WiMAX PHY layer, the QoS provision is perhaps one of the most intriguing features defined in the WiMAX MAC layer. Due to the association of an SFID with a CID, every packet arriving in the WiMAX network is related to a set of QoS parameters supported by the WiMAX scheduler. WiMAX defines five types of uplink scheduling services: unsolicited grant service (UGS), real-time polling service

(rtPS), extended real-time polling service (ertPS), non-real-time polling service (nrtPS) and best effort (BE) service.

The UGS is granted a fixed bandwidth allocation for a data stream that consists of a constant data packet generation in periodic intervals. This service type is suitable for real-time applications such as voice over IP (VoIP). The rtPS is issued with periodic transmission opportunities for bandwidth requests. This type of services is ideal for real-time variable bit rate traffic such as video and audio streaming. A newly defined scheduling type in 802.16e-2005 is ertPS, which combines the traits of UGS and rtPS. The ertPS scheduling type is offered with unsolicited bandwidth grants but at a variable bandwidth. This aims to support data streams that have variable size data in a periodic interval such as VoIP with silence suppression.

The nrtPS scheduling type is appropriate for delay-tolerant applications, such as FTP, but requires a minimum service rate. This scheduling service uses contention or unicast request opportunities to issue requests for bandwidth. Finally, the BE scheduling type is served based on demand basis, and it is not provided with bandwidth reservation or any QoS provision. Table 2-2 summarizes the properties and the QoS parameters associated with each type of uplink scheduling service in the WiMAX MAC layer.

Table 2-2: A summary of types of scheduling services in the WiMAX MAC layer

Scheduling	Traffic Traits	Applications	QoS Specifications
UGS	Real-time periodic constant bit rate	VoIP	Max sustained traffic rate Max latency, Tolerated jitter Unsolicited grant interval
rtPS	Real-time periodic variable bit rate	Video/audio streaming	Min reserved traffic rate Max sustained traffic rate Max latency Unsolicited polling interval
ertPS	Real-time periodic variable bit rate	VoIP with silence suppression	Min. reserved traffic rate Max. sustained traffic rate Max latency, Tolerated jitter Unsolicited grant interval
nrtPS	Non-real-time variable bit rate	FTP	Max sustained traffic rate Min reserved traffic rate
BE	No QoS requirements	Web browsing	Max sustained traffic rate

2.2.2.2.3 Bandwidth Allocation and Request (802.16 Standard: 6.3.6)

In UL, when a SS has data to send, it generates requests to inform the BS the amount of bandwidth it requires. The request is specified in number of bytes, including the MAC header and payload, and it is polled by the BS at designated time. When a poll is unicast, the receiving SS is directly allocated with sufficient

bandwidth to make a request. In contrast, if a poll is broadcast or multicast to a group of subscriber stations, an SS belonging to the polled group contents for an opportunity to send the request. Based on the QoS specification of the connection and resource availability, the BS determines which request is accepted, and generates a grant. A grant is a burst profile embedded in the UL-MAP, indicating the bandwidth boundaries allocated to the SS in a subframe. Bandwidth requests are constructed on a per connection basis, but the grants are issued on a per SS basis. Figure 2.7 illustrates the procedure of an rtPS or an nrtPS connection requesting for bandwidth for data transmission.

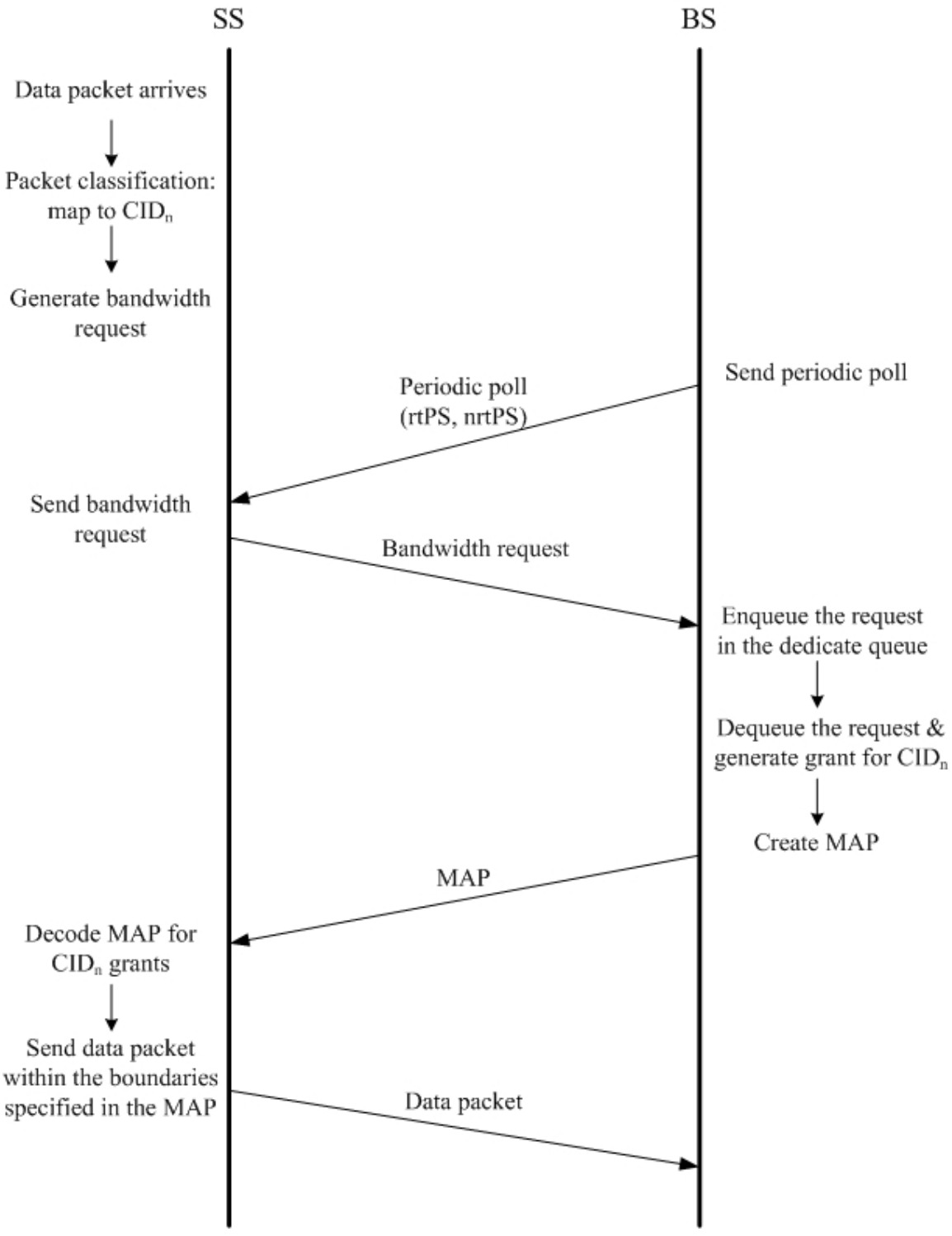

Figure 2.7: The bandwidth request mechanism of an rtPS or an nrtPS connection in the UL direction

2.2.2.2.4 HARQ Option

The hybrid ARQ (HARQ) is an option defined in the WiMAX MAC common part sublayer, but it is only supported under the OFDMA PHY profile. HARQ is an error control mechanism that is based on the stop-and-wait protocol. HARQ enhances the performance of a connection in a poor channel condition at the cost of throughput reduction. A corrupted frame is stored and retransmitted. Upon the retransmission, the receiving node combines the retransmitted and previously stored corrupted packet to attain a better signal-to-noise (SNR) and coding gain. The HARQ mechanism can be enabled on a per CID basis.

This ends the material in WiMAX that is included in this thesis. This chapter has provided background on TCP and WiMAX. The next chapter will focus on presenting the proposed cross-layer technique involving TCP and the WiMAX MAC layer.

CHAPTER 3: THE PROPOSED CROSS-LAYER TECHNIQUE: THE ALGORITHM, IMPLEMENTATIONS AND ANALYTICAL MODEL

As described in Chapter 2, TCP is one of the host layers that reside on end-hosts, and the MAC layer is one of the media layers, which operate at the network core. TCP acts in the fundamental role of controlling the end-to-end data transport, and injecting data into the network at variable rates, depending on its assessments of the network condition over the entire transmission path. Nevertheless, TCP is logically further from the physical transport medium than the MAC layer; therefore, it is not capable of adapting well to a rapidly changing wireless link. The MAC layer, which is the bottommost sublayer of the link layer, is logically situated directly above the physical medium, and it delivers data across the wireless link. Though TCP and the MAC layer operate concurrently over the same network condition, they have different perspectives. Instead of operating independently of each other, it is more beneficial to allow communication between the two layers to achieve a better comprehension of the network state.

3.1 Algorithm Overview

TCP utilizes many mechanisms, such as duplicate ACKs and RTT estimations, in an attempt to infer the network condition. The key parameter that reflects the consequences of these inferences is the size of congestion window.

This parameter, *cwnd*, regulates the sending rate of TCP segments; in other words, it is an implicit indication of how a TCP connection perceives the state of network congestion over the entire data transmission path. More significantly, it determines the quantity and the rate of packets arriving at the MAC layer. Since the wireless band is a scarce resource, it is critical that the scheduler optimizes its resource allocation to the desired connections. The *cwnd* provides information on the packet arrival rate at the MAC layer, and a complementary aspect of the network condition. Thus, by incorporating the *cwnd* parameter into the MAC layer, it is capable of allocating its resources in a more intelligent fashion.

At the WiMAX MAC layer, a connection with a scheduling service type of UGS, rtPS, ertPS, or nrtPS is assigned a dedicated queue, and the queue is associated with a specific set of QoS parameters as described in Section 2.2.2. One of the universally adopted scheduling schemes is Weighted Fair Queuing (WFQ) [30], [31], which has also been implemented in the OPNET WiMAX model. Depending on the QoS specification, the weight assigned to each queue can be different. This weight value essentially determines the amount of resources that the MAC scheduler agrees to allocate to the queue.

The weight of a queue is resolved according to the QoS requirement of the queue at the time of admission, and it remains fixed throughout the connection session. However, the MAC scheduler should be able to adapt the manner in which it distributes its resources. Thus, I propose that the weight of each queue should fluctuate with respect to the *cwnd* values from TCP. More specifically, I propose that the weight of each queue should vary according to

Equation 3.1, where *W* represents the original weight assigned to the queue at admission time, *c* denotes the *cwnd* values of the TCP flow associated with the queue, and *a* is the coefficient of the weight-adjusting factor. The subscript *n* denotes the n^{th} queue, and the subscript *t* represents the total of a property of all *N* queues in the WiMAX network.

$$W_n^{'} = W_n + a\frac{c_n}{c_t}W_n, \quad where\ c_t = \sum_i^N c_i \tag{3.1}$$

The resulting weight is a value that is reflective of the network congestion condition as perceived by a TCP flow. It grants a queue an extra portion of its original weight, and the proportion is determined by comparing the *cwnd* value of the queue to that of other queues. In other words, a queue is given more resources if the *cwnd* value associated with the queue is high, as compared to other queues. Nevertheless, the extra portion is multiplied by the original weight, in an effort to reinforce the QoS provisions that were originally promised by the scheduler.

Therefore, the proposed algorithm still sustains differentiation of QoS across queues, and at the same time reflects resource distribution in accordance with the network condition. Within limits, the algorithm favours queues with fair channel conditions over the entire transmission path, by granting them extra bandwidth. Finally, the coefficient of the weight-adjusting factor is itself an adjustable factor that can be manipulated by system designers in order to establish a more aggressive system. In brief, the weight of the queue is adaptive according to the *cwnd* value of TCP.

3.1.1 Discussion of Extreme Cases and Limitations of the Proposed Scheme

The weight-adjusting factor, c_n/c_t, is defined as the ratio of the *cwnd* value of the n^{th} queue to the sum of that of all N queues in the network. Thus, it can never be grater than one. As a result, the maximum value of W_n' is limited at $(1+a)W_n$, and the minimum value of W_n' remains as W_n. Consequently, a connection cannot monopolize the bandwidth consumption, regardless of its fair channel condition and the resulting high *cwnd* values. In contrast, a connection is still provided with a minimum weight of W_n even if it suffers from bad channel conditions. The proposed scheme maintains fairness across the queues, in the sense that the resource granted to a data flow is bounded by a minimum and a maximum that are both determined based on the original queue weight, W_n.

Assuming a total of N TCP flows (i.e. N queues) in the WiMAX network, the total queue weight admitted to the system is W_t as described in Equation 3.2. In the proposed scheme, the weight granted to each queue is slightly more than the original, due to the second term of Equation 3.1. However, the maximum of total weight granted to the queues in the proposed scheme is bounded as described in Equation 3.3. The proposed algorithm, though more aggressive than the original scheme by the virtue of assigning fair quality channels with extra bandwidth, is still a stable system. The issue of granting of more resources than originally admitted can be resolved by a conservative admission control algorithm, such that it considers Equation 3.3 when admitting a queue.

$$W_{t,old} = \sum_{i}^{N} W_i \tag{3.2}$$

$$\max\left(W_{t,new}\right) = \sum_{i}^{N} W_i + a \cdot \max_i\left(W_i\right) \tag{3.3}$$

One of the constraints of the proposed scheme is its application to TCP connections only. A transport mechanism such as UDP, which maintains no estimations of the network condition, cannot utilize this scheme. Another constraint of the proposed scheme occurs in the situation where a queue accumulates packets from more than one TCP flow. Extraction of *cwnd* values from different data flows may result in the proposed scheme facing problematic discernment of the network condition. More specifically, the *cwnd* value extracted from a packet of a particular TCP data flow of a queue may be mistakenly interpreted as the congestion assessment made by another TCP data flow of the same queue. However, this confusion can be resolved by extracting additional information such as the source/destination IP addresses, and source/destination port numbers from the header field of a TCP/IP datagram.

3.2 Design Modification of the TCP Segment Format

In order to transport the value of *cwnd* to the MAC layer, I utilized a portion of the option field of a standard TCP segment. I denote the name of the field *Cwnd Option*, and it is currently set to occupy 32 bits. I embed the *cwnd* value in the Cwnd Option field, instead of utilizing explicit messages or an independent protocol such as the Internet Control Message Protocol (ICMP). A scaled *cwnd* value approach can be considered if a smaller option field size is desirable.

Figure 3.1 illustrates a typical TCP header with the newly introduced Cwnd Option field.

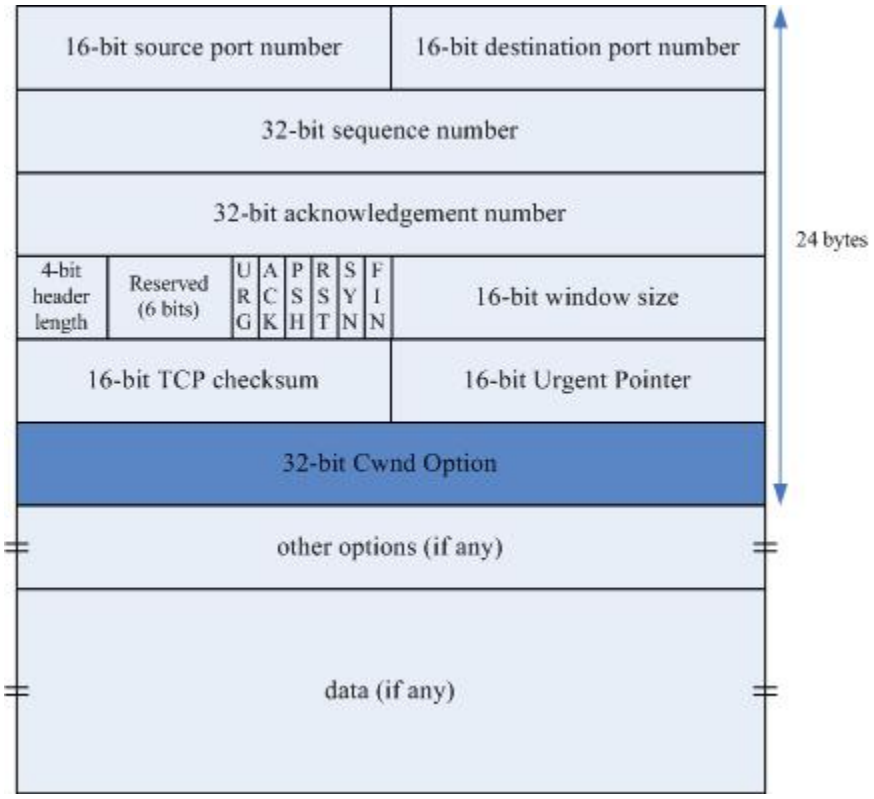

Figure 3.1: The new TCP segment format

3.3 Design Modifications of the WiMAX MAC Layer Operation

Once a packet arrives at a MAC queue, the embedded *cwnd* value is extracted from the header, and the weight of the queue is calculated based on the previously described Equation 3.1. In order to maintain the bandwidth usage of a queue, the Modified Deficit Round Robin (MDRR) [20] queue service discipline is employed at the downlink transmission of the BS. The proposed scheme retains the structure of the MDRR queue service discipline, and it is implemented as a modified version of the MDRR queue service discipline in the downlink scheduler.

A queue in an MDRR discipline is defined by two parameters, a weight value and a deficit counter. When a queue is being served by the scheduler, the weight value indicates the maximum amount of data that a queue is allowed to *dequeue* in one round. The deficit counter accumulates the amount of bandwidth that has been used up by the queue. The deficit counter is initialized to the weight value of the queue, and is deducted by the size of the packet that is being serviced by the scheduler. The scheduler continues to serve a non-empty queue until its deficit counter reaches zero or below, and then moves on to the next queue in a round-robin fashion. The deficit counter is replenished with the weight of the queue when the scheduler loops back to serve the same queue in the next round. This scheduling discipline ensures that every queue is guaranteed a visit by the scheduler in one round, and at the same time, it provides differentiation in QoS.

In my implementation, I retain the fundamental structure of the MDRR queue service discipline, but I replenish the deficit counter with a custom-calculated queue weight as described in Equation 3.1. In addition to the cwnd-dependent weight value (i.e. W_n'), I modify the scheduling discipline to be slightly more aggressive. The scheduler continues to serve a queue if the number of packets left in the queue is one, regardless the value of the deficit counter. The idea of this strategy is to empty a queue when possible, instead of leaving one packet in the queue waiting to be served in the next round. Figure 3.2 illustrates the logic of the MDRR queue service discipline, and the gray boxes indicate changes I made in addition to the original design.

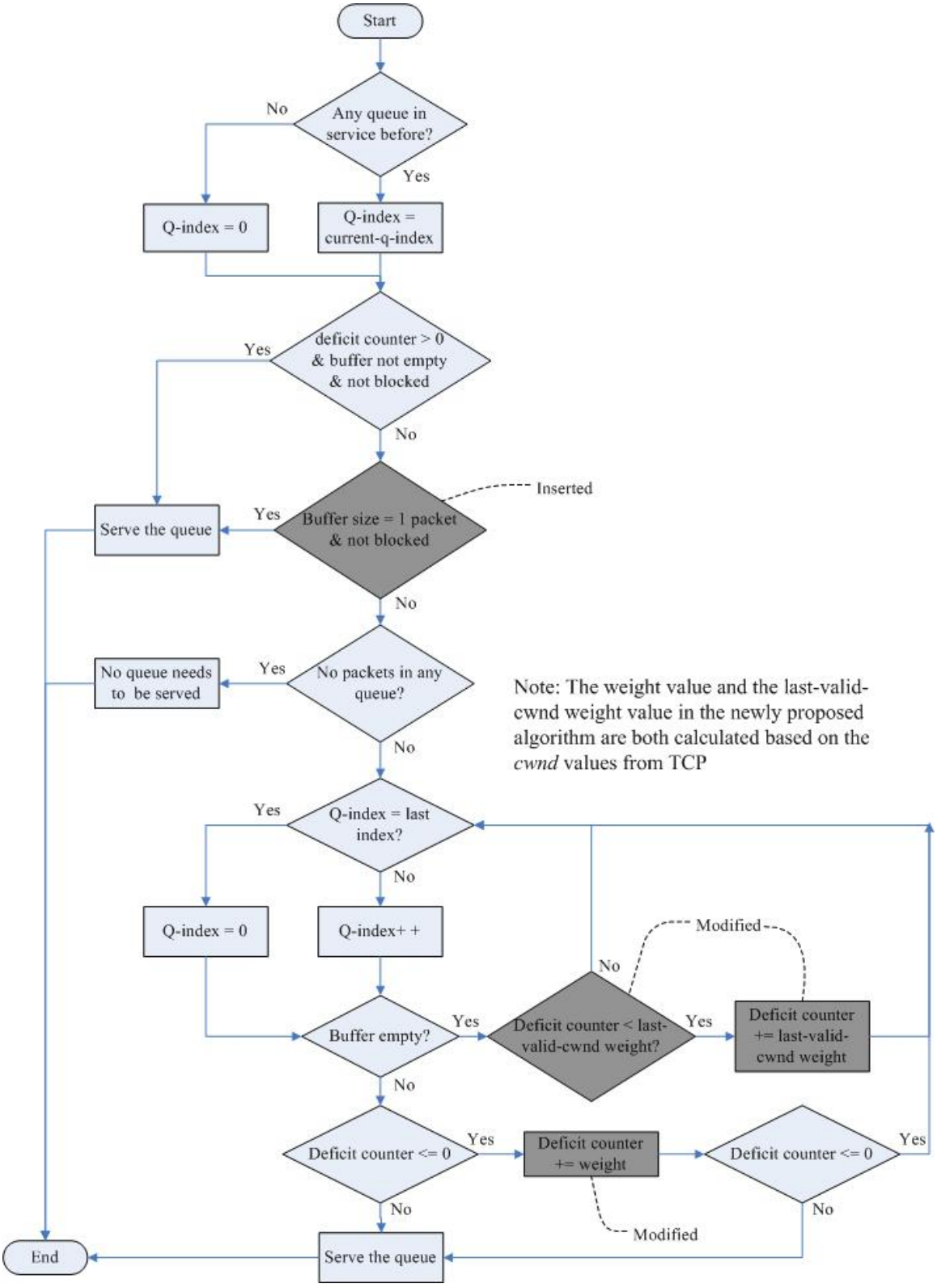

Figure 3.2: The flowchart of the MDRR queue service discipline and indications on modifications made

As illustrated in the flowchart, the deficit counter in my implementation is updated with the two different queue weights, depending on the buffer status. If a queue is non-empty, the regular cwnd-dependent weight value, which is calculated based on the current *cwnd* value, is utilized to replenish the deficit counter. However, when a queue is empty, the *cwnd* value would be zero due to no packet being present in the queue, which results in a smaller cwnd-dependent weight value. The observation of a zero *cwnd* value misleads the scheduler to arrive at an inaccurate interpretation on the state of a data flow, where an empty queue can be due to slow data generation instead of a severe channel condition. Under such a circumstance, a queue should not lose its achieved status because of the temporary empty state. To address this problem, I retain a copy of the *cwnd* value of the last packet existed in the queue, and I calculate the cwnd-dependent weight based on this *last-valid cwnd value*. Hence, the *last-valid-cwnd weight* value is utilized when the scheduler refreshes the deficit counter of an empty queue.

3.4 The Analytical Model of the Algorithm

To understand the effect of the proposed algorithm on the MAC scheduling scheme, analysis of the queue service rate and queue delay was conducted. Consider a WiMAX network with a total of *N* queues, each of which is associated with an originally assigned weight value of *W*, a congestion window size of *c*, a cwnd-dependent weight value of *W'*, and a queue size of *Q*. Figure 3.3 illustrates the concept of the queue system that is utilized to develop the analytical model.

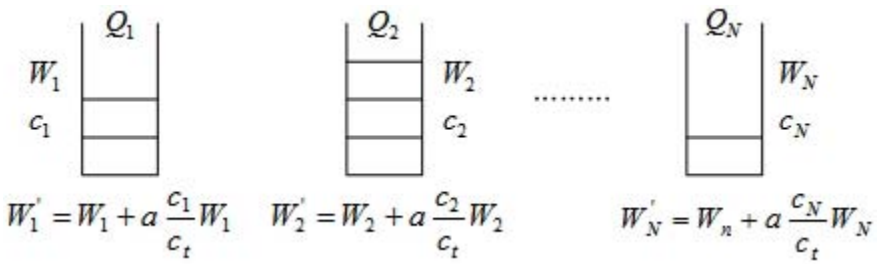

Figure 3.3: An illustration of the MAC queue concept

3.4.1 The Analysis of Queue Service Rate

The service rate is defined as the amount of data served in a window of time. In this case, I define the service rate of a queue be the amount of data served in a queue in one round, and one round corresponds to the amount of time that is required for the scheduler to loop back to the same queue. Equation 3.4 describes the definition of the queue service rate, and the related notations.

$$\mu_n = service\ rate\ of\ \ n^{th} queue = \frac{data\ served\ in\ n^{th}\ queue\ in\ one\ round}{scheduler\ wrap\ around\ time} = \frac{d_n}{T} \qquad (3.4)$$

The amount of data served in each queue in one round of scheduling is different under three conditions:

a) All Q_n packets are scheduled
b) The queue reaches its allowed scheduling limit W_n'
c) Only a residual bandwidth, B_r, is left available to schedule a portion of data in the queue.

As a result, the amount of data served in a queue in one round is Q_n, W_n', or B_r. Case (a) is a condition that happens when Q_n is smaller than W_n', which implies that the traffic of the queue is light. Case (b) occurs when packets start to accumulate with increasing number of stations in the network or *cwnd* values.

The queue size eventually becomes greater than the queue weight, and the amount of data served in one round is limited to W_n'. Case (c) is a situation that occurs when a queue is the last queue to be served in a subframe time, and only residual bandwidth is left available for scheduling. However, it is unlikely that a queue is always the last queue to be scheduled in every downlink subframe. Therefore, case (b) can be deemed as the most common case of all three, and it should be a legitimate representation of the three on the amount of data being served in one round of scheduling.

After determining the amount of data served per round, the next task is to formulate the wrap-around time of the scheduler. The time, *T*, required for the scheduler to loop through all queues is dependent on the number of queues, the traffic intensity of the queues, and the bandwidth capacity of the system. The wrap-around time increases as the number and sizes of queues increase, but shortens if the network bandwidth capacity is large. This leads to an expression of the wrap-around time as described in Equation 3.5, where *B* indicates the bandwidth capacity of the system, and τ is denoted to represent the time length of a WiMAX MAC frame, including the preamble and transition time gaps.

$$\textit{Wrap around time} \quad T = \frac{\sum_{i}^{N} \min\left(W_i^{'}, Q_i\right)}{B} \cdot \tau \tag{3.5}$$

The equation indicates that the wrap-around time is only a fraction of a frame time if the number of queues is few, or the traffic is light (i.e. queue sizes are small). The scheduler is capable of looping through all queues multiple times

in a frame time. Whereas, if the number of stations is large, or the traffic intensity of the queues are high, such that the network capacity cannot accommodate all data at once, the scheduler requires more than one frame time to revisit the same queue.

Combining the amount of data served per round and the wrap-around time, the queue service rate of each case is derived as illustrated in Equation 3.6 to 3.8. Note that the minimum of W_i' and Q_i is always Q_i in case (a) since case (a) occurs when the traffic is light. In contrast, the minimum of W_i' and Q_i is W_i' in case (b) and (c).

$$case\ (a) \quad \mu_n = \frac{B}{\tau} \cdot \frac{Q_n}{\sum_i^N \min\left(W_i', Q_i\right)} = \frac{B}{\tau} \cdot \frac{Q_n}{\sum_i^N Q_i} \tag{3.6}$$

$$case\ (b) \quad \mu_n = \frac{B}{\tau} \cdot \frac{W_n'}{\sum_i^N \min\left(W_i', Q_i\right)} = \frac{B}{\tau} \cdot \frac{W_n'}{\sum_i^N W_i'} \tag{3.7}$$

$$case\ (c) \quad \mu_n = \frac{B}{\tau} \cdot \frac{B_r}{\sum_i^N \min\left(W_i', Q_i\right)} = \frac{B}{\tau} \cdot \frac{B_r}{\sum_i^N W_i'} \tag{3.8}$$

The service rate of case (a) is dependent on the queue size, and it implies that packets arriving at the queue are all served in one round, which is reasonable when the traffic is light. In contrast, the service rate is regulated by the weight of a queue in case (b). In case (c), the service rate depends on B_r but always smaller than case (b) (i.e. $B_r < W_n'$). In the original algorithm, the queue weight is constant (i.e. a=0) throughout the transmission session; however, the queue weight fluctuates with *cwnd* values in the proposed design. Assuming that

the original weight of all queues are equal (i.e. $W_i = W_n$ for all i), and substituting the weight expression of the original and proposed design, the service rates of case (b) and (c) can be rewritten as in Equation 3.9 to 3.12.

$$case\ (b),\ original \qquad \mu_{n,old} = \frac{B}{\tau} \cdot \frac{W_n^{'}}{\sum_{i}^{N} W_i^{'}} = \frac{B}{\tau} \cdot \frac{W_n}{N W_n} = \frac{B}{\tau} \cdot \frac{1}{N} \qquad (3.9)$$

$$case\ (b),\ proposed \qquad \mu_{n,new} = \frac{B}{\tau} \cdot \frac{W_n^{'}}{\sum_{i}^{N} W_i^{'}} = \frac{B}{\tau} \cdot \frac{W_n + a\frac{c_n}{c_t} W_n}{N W_n + a W_n} = \frac{B}{\tau} \cdot \frac{1 + a\frac{c_n}{c_t}}{N + a} \qquad (3.10)$$

$$case\ (c),\ original \qquad \mu_{n,old} = \frac{B}{\tau} \cdot \frac{B_r}{\sum_{i}^{N} W_i^{'}} = \frac{B}{\tau} \cdot \frac{B_r}{N W_n} \qquad (3.11)$$

$$case\ (c),\ proposed \qquad \mu_{n,new} = \frac{B}{\tau} \cdot \frac{B_r}{\sum_{i}^{N} W_i^{'}} = \frac{B}{\tau} \cdot \frac{B_r}{N W_n + a W_n} = \frac{B}{\tau} \cdot \frac{B_r}{(N + a) W_n} \qquad (3.12)$$

Comparing the queue service rates of the newly proposed and original algorithm, comparison equations are resolved as shown in Equation 3.13 and 3.14 for case (b) and (c).

$$case\ (b),\ comparison\ equation \qquad \frac{\mu_{n,new}}{\mu_{n,old}} = \frac{N}{N + a}\left(1 + a\frac{c_n}{c_t}\right) \qquad (3.13)$$

$$case\ (c),\ comparison\ equation \qquad \frac{\mu_{n,new}}{\mu_{n,old}} = \frac{N}{N + a} \qquad (3.14)$$

The comparison equations indicate that the number of queues, N, needs to be large in order for the new algorithm to reduce the effect of the weight-adjusting factor coefficient in the denominator. Despite that, the service rate in

case (b) is improved if the term in the bracket of Equation 3.13 can outweigh the $N/(N+a)$ factor. On the other hand, if the *cwnd* ratio (i.e. c_n/c_t) is poor, the service rate of the proposed scheme can be lower than in the original. This is expected since the scheduler attempts to allocate resources based on the network condition assessments. As a result, for a queue with bad-channel condition, the resources supplied to the queue can be less. Furthermore, a large coefficient, a, results in a more aggressive resource allocation scheme, but it requires the number of queues to be even larger, in order for the proposed scheme to deliver a better queue service rate than the original one.

Without the *cwnd* ratio term in case (c), the proposed scheme performs worse than the original because the sum of the cwnd-dependent weight, W_n', of all queues is virtually equivalent to admitting an additional queue into the network, as implied in Equation 3.3. The service rate of a queue in the proposed scheme relies on c_n/c_t and a to outperform the original design.

Furthermore, consider the coefficient, a, is fixed in Equation 3.13, the proposed design performs better with increasing N. However, when N is large such that the $N/(N+a)$ factor approaches unity, the gain in performance is bounded by a limit given by $[1 + a(c_n/c_t)]$. In addition, consider Equation 3.10, if N is sufficiently larger than a, the $(N+a)$ term in the denominator is dominated by N. As a result, the service rate increases with a. However, if a continues to grow such that a is equally influential as N in the denominator, elevation in a at the same time dilutes the gain in the numerator. Consequently, the performance of

the proposed scheme increases with *N* and *a*, but the gains are bounded at certain limits.

3.4.1.1 The Expected Value of Queue Service Rate

As previously described, the queue service rates are differentiated into three occurrences, with case (b) being the most common case. Therefore, the expected value of queue service rate derived in this sub-section focuses only on case (b) (i.e. Equation 3.7). The expected value of the queue service rate in case (b) is determined by the expected value of the queue weight and the sum of queue weights. The queue weight of the original algorithm is constant while that of the proposed algorithm fluctuates depending on *cwnd* values. Thus, the expected values of queue weight of the original and proposed algorithms are different as illustrated in Equation 3.15 and 3.16.

$$\begin{aligned} original \quad & W_{n,old} = W_n \\ & E\left[W_{n,old}\right] = W_n \end{aligned} \tag{3.15}$$

$$\begin{aligned} proposed \quad & W_{n,new} = W_n + a\frac{c_n}{c_t}W_n \\ & E\left[W_{n,new}\right] = E\left[W_n + a\frac{c_n}{c_t}W_n\right] = W_n + a \cdot E\left[\frac{c_n}{c_t}\right] \cdot W_n \end{aligned} \tag{3.16}$$

From Equation 3.16, the expected value of the queue weight of the proposed scheme is dependent on the expected value of *cwnd* ratio. To determine the expected *cwnd* ratio, I consider the worst and best possible cases of a *cwnd* ratio.

Denoting c_{max} as the maximum value of *cwnd*, the smallest *cwnd* ratio of the nth queue occurs when c_n is equal to one while the congestion windows of all other queues are equal to c_{max}. In contrast, the largest *cwnd* ratio of the nth queue is c_n equals to c_{max}, and the congestion windows of all other queues equal to one. The mathematical representation of the smallest and largest *cwnd* ratios are expressed in Equation 3.17 and 3.18. The condition of c_{max} being sufficiently larger than N is valid since the congestion window size is maintained in bytes, and the maximum can be in the range of thousands or tens of thousands bytes.

$$\left(\frac{c_n}{c_t}\right)_{smallest} = \frac{1}{(N-1)c_{\max}+1} \cong \frac{1}{(N-1)\cdot c_{\max}} \tag{3.17}$$

$$\left(\frac{c_n}{c_t}\right)_{largest} = \frac{c_{\max}}{(N-1)+c_{\max}} \cong \frac{c_{\max}}{c_{\max}} = 1 \qquad if\ c_{\max} >> N \tag{3.18}$$

The distribution of the *cwnd* ratio can vary across queues. For queues that constantly enjoy fair channel conditions, their *cwnd* ratios are likely to be distributed near the large values range. On the other hand, if a station continuously suffers from bad channel conditions, its *cwnd* ratios are likely to be concentrated at the low values range. Consequently, the distribution of the *cwnd* ratio of a queue is highly dependent on the physical channel condition of the queue. Since the channel condition is equally likely to spread between good and bad states, I assume the *cwnd* ratio to be uniformly distributed between the smallest and largest values. Thus, the expected value of *cwnd* ratio is calculated as in Equation 3.19. Substituting the expected *cwnd* ratio back to Equation 3.16,

the expected value of queue weight of the proposed scheme is shown in Equation 3.20.

$$E\left[\frac{c_n}{c_t}\right] = \frac{1}{2}\cdot\left(\left(\frac{c_n}{c_t}\right)_{smallest} + \left(\frac{c_n}{c_t}\right)_{largest}\right)$$
$$\cong \frac{1}{2}\cdot\left(\frac{1}{(N-1)\cdot c_{\max}} + 1\right) \approx \frac{1}{2}(0+1) = \frac{1}{2} \qquad as\ c_{\max} \to \infty \tag{3.19}$$

$$E\left[W_{n,new}\right] = W_n + a\cdot E\left[\frac{c_n}{c_t}\right]\cdot W_n = W_n + \frac{a}{2}W_n \tag{3.20}$$

The next term to be resolved is the expected value of the sum of queue weights. Given that the minimum of W_n' and Q_n is W_n' in case (b), and assuming the original weights of all queues are equal in the network (i.e. $W_i = W_n$ for all i), the expected values of the sum of queue weights are presented in Equation 3.21 and 3.22.

$$E\left[\sum_i^N W_{i,old}\right] = E\left[N\cdot W_n\right] = N\cdot W_n \tag{3.21}$$

$$E\left[\sum_i^N W_{i,new}\right] = E\left[\sum_i^N W_i + a\frac{c_i}{c_t}W_i\right]$$
$$= E\left[\sum_i^N W_i\right] + a\cdot E\left[\sum_i^N \frac{c_i}{c_t}W_i\right] = N\cdot W_n + a\cdot W_n = (N+a)W_n \tag{3.22}$$

Combining the expected values of queue weight and the sum of the weights, the average queue service rates for both the original and proposed scheme can be found in Equation 3.23 and 3.24.

$$E\left[\mu_{n,old}\right] = \frac{B}{\tau} \cdot \frac{E\left[W_{n,old}\right]}{E\left[\sum_{i}^{N} W_{i,old}\right]} = \frac{B}{\tau} \cdot \frac{W_n}{N \cdot W_n} = \frac{B}{\tau} \cdot \frac{1}{N} \tag{3.23}$$

$$E\left[\mu_{n,new}\right] = \frac{B}{\tau} \cdot \frac{E\left[W_{n,new}\right]}{E\left[\sum_{i}^{N} W_{i,new}\right]} = \frac{B}{\tau} \cdot \frac{W_n + \frac{a}{2} W_n}{(N+a) \cdot W_n} = \frac{B}{\tau} \cdot \frac{1+\frac{a}{2}}{N+a} \tag{3.24}$$

The average service rate decreases as *N* increases in both designs, which is reasonable since resources are shared by more clients. However, for the proposed scheme, the service rate does not decrease as fast as the original, and it decreases even slower as *a* is larger. Comparing the average service rate of the proposed to original scheme, a comparison equation can be formulated as shown in Equation 3.25. In addition, Figure 3.4 illustrates the comparison equation with respect to increasing values of *N* while *a* is fixed in each plot.

$$\frac{E\left[\mu_{n,new}\right]}{E\left[\mu_{n,old}\right]} = \frac{N}{N+a} \cdot \left(1+\frac{a}{2}\right) \tag{3.25}$$

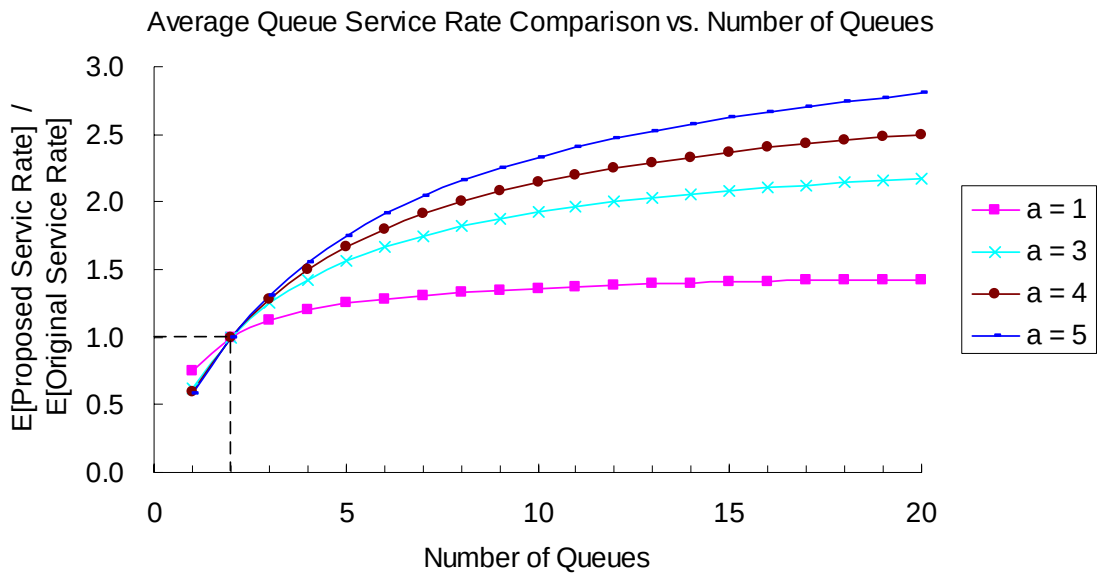

Figure 3.4: Comparison of the average queue service rate between the proposed and original scheme with changing values of *N* while *a* is fixed

The figure depicts that all plots intersect at the point, where *N* equals to two, and the comparison ratio equals to one. In other words, when the number of queues in the network is two, the proposed scheme performs equally well as the original, independent of the value of the weight-adjusting factor coefficient. However, the proposed scheme shows its advantage as the number of queues continues to grow, and the advantage is even more evident as the value of the coefficient increases.

The graph also indicates that the gain in the proposed scheme with respect to *N* is bounded at a certain limit when *N* is large. This observation is obvious in the plot of *a*=1, where the plot is close to horizontal when *N* is large. Furthermore, the gain between each value of *a* (i.e. the vertical gaps between each plot) reduces with increasing *a*. This observation suggests that the gain with respect to *a* is also limited to a certain bound. These two observations confirm

the conclusions drawn at the end of the derivation of queue service rate. The comparison equation is plotted with respect to increasing values of *a* while *N* is fixed in each plot, as illustrated in Figure 3.5.

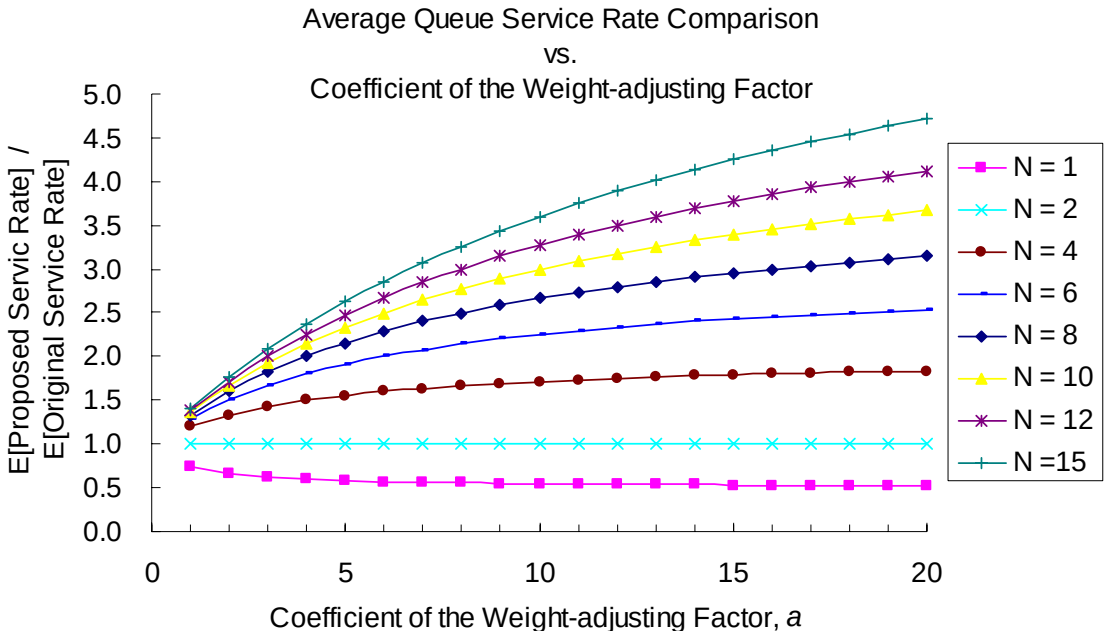

Figure 3.5: Comparison of the average queue service rate between the proposed and original scheme with changing values of *a* while *N* is fixed

The queue service rate of the proposed design is worse than the original when *N* is one, independent of the values of coefficients. The advantage of the proposed design begins to show when *N* is greater than two. For the same *N*, the advantage grows with respect to increasing *a*, but the gain approaches to a constant when *a* is too large compared to *N*. These observations comply with the observations made in Figure 3.4. Furthermore, Figure 3.5 demonstrates that the advantage of large *a* is more evident when *N* is large. In other words, large values of *a* require the number of queues in the network be sufficiently large to deliver a more evident performance gain, and if *N* is small, a small value of *a* is sufficient.

3.4.2 The Analysis of Queue Delay

The approach taken in the analysis of queue delay is to first develop an expression for queue size, and then apply Little's formula after evaluating the average queue size to achieve the average queue delay.

The size of a queue varies as packets arrive at the queue, and as they are served leaving the queue. Thus, the queue size is related to the arrival rate, λ, and the service rate, μ, as described in Equation 3.26, where *q* is denoted to represent the difference between the two. Note that the expression, *(t)*, signifies the time-varying characteristic of each term.

$$q(t) = \lambda(t) - \mu(t) \tag{3.26}$$

However, the discrepancy between the arrival and service rate represents only the changes in queue size, instead of the actual queue size. If a queue is initially empty, the actual queue size, *Q*, can be obtained by integrating the changes in queue size over time as illustrated in Equation 3.27. Taking an integral over time is equivalent to multiplying by a window of time, Δt, in the discrete form, as shown in Equation 3.28. Note that changes in queue size can be negative; nevertheless, the queue size is subject to a lower bound of zero, and an upper bound of the buffer size.

$$Q = \int_{t_0} q(t)\, dt = \int_{t_0} \lambda(t) - \mu(t)\, dt \tag{3.27}$$

$$Q = q(t) \cdot \Delta t = \left(\lambda(t) - \mu(t)\right) \cdot \Delta t \tag{3.28}$$

The average queue size is obtained by evaluating the expected value, as shown in Equation 3.29. For simplicity, the remainder of this sub-section will focus only on the derivation of the discrete form.

$$E[Q] = E\left[\lambda(t) - \mu(t)\right] \cdot \Delta t = \left(E\left[\lambda(t)\right] - E\left[\mu(t)\right]\right) \cdot \Delta t \tag{3.29}$$

The equation complies with the fact that if the average arrival rate is greater than the average service rate in a window of time, the queue size grows as the time window stretches longer. In contrast, if the average arrival rate is smaller than the average service rate, the queue size shrinks over time. However, the overall service rate can never be greater than the overall arrival rate because the service rate diminishes to zero when no packet exists in the queue. This logical constraint ensures that the average queue size is never negative.

With an expression for the average queue size, the expected queue delay, *d*, can be derived after applying the Little's formula described in Section 3.2.1 of [21]. The Little's formula states that the average queue size is the product of the average net arrival rate and the average queue delay, as shown in Equation 3.30. An expression for the average queue delay of the scheme arises from the Little's formula, as described in Equation 3.31. For simplicity, the expression of *(t)* is dropped in the following equations.

$$Little's\ Formula: \quad E[Q] = net\ average\ arrival\ rate \times E[d] \tag{3.30}$$

$$E[d] = \frac{E[Q]}{net\ arrival\ rate} = \frac{(E[\lambda] - E[\mu]) \cdot \Delta t}{E[\lambda]} = \left(1 - \frac{E[\mu]}{E[\lambda]}\right) \cdot \Delta t \tag{3.31}$$

The expected value of queue delay is subject to a lower bound of zero because the average queue size is never negative. In addition, since λ and μ are never negative, the equation suggests that the maximum expected queue delay never exceeds Δt. This is intuitively incorrect because the queue delay should be able to grow to infinity if the service rate is zero. Nevertheless, this upper bound of Δt on the queue delay becomes sensible since the average queue size is estimated at $t_0+\Delta t$ seconds; therefore, the maximum delay that a packet can experience from t_0 to $t_0+\Delta t$ second is Δt seconds.

From this perspective, the terms in the bracket of Equation 3.31 can be interpreted as the fraction of the window of time, Δt, that a packet experiences as queue delay. Since the window of time is arbitrary, it can be replaced by the maximum amount of time that a packet can spend in a queue. Assuming the service rate is non-zero, the maximum queue delay occurs when a packet is accepted into a fully occupied queue. Denoting the buffer size of a queue as F, the maximum queue delay, d_{max}, can be expressed as in Equation 3.32, and its expected value in Equation 3.33.

$$d_{\max} = service\ time\ \ per\ unit\ \ size \times buffer\ \ size = \frac{1}{\mu} \cdot F \tag{3.32}$$

$$E[d_{\max}] = \frac{1}{E[\mu]} \cdot F \tag{3.33}$$

The actual delay emerges by replacing the arbitrary window of time with d_{max}. Denoting the actual queue delay to D, the average queue delay can be written as in Equation 3.34.

$$\begin{aligned} E[D] &= \left(1 - \frac{E[\mu(t)]}{E[\lambda(t)]}\right) \cdot E[d_{\max}] \\ &= \left(1 - \frac{E[\mu(t)]}{E[\lambda(t)]}\right) \cdot \left(\frac{1}{E[\mu(t)]} \cdot F\right) = \left(\frac{1}{E[\mu(t)]} - \frac{1}{E[\lambda(t)]}\right) \cdot F \end{aligned} \tag{3.34}$$

The expected queue delay reduces as the service rate increases, or as the arrival rate decreases. However, the expected queue delay never falls below zero because the overall service rate of a queue cannot be greater than the overall arrival rate. Furthermore, if the arrival rate is large, the queue delay expression simplifies to d_{max}, which complies with the fact that a queue is always full when the arrival rate is infinite.

The average service rate of a queue was derived in Section 3.4.1.1, and the arrival rate of a MAC queue is approximately equivalent to the sending rate of its corresponding TCP flow. Assuming the average arrival rates from TCP flows are identical for each queue, the queue delay depends only on the queue service rate to differentiate the performance between the original and proposed algorithm. Substituting the average service rate into the equation of average queue delay,

the queue delays of the original and proposed designs are described in Equation 3.35 and 3.36 respectively.

$$E[D_{old}] = \left(\frac{1}{E[\mu_{old}]} - \frac{1}{E[\lambda]} \right) \cdot F = \left(\frac{\tau}{B} N - \frac{1}{E[\lambda]} \right) \cdot F \tag{3.35}$$

$$E[D_{new}] = \left(\frac{1}{E[\mu_{new}]} - \frac{1}{E[\lambda]} \right) \cdot F = \left(\frac{\tau}{B} \frac{N+a}{1+a/2} - \frac{1}{E[\lambda]} \right) \cdot F \tag{3.36}$$

The queue delay of the proposed scheme is smaller than the original if the service rate of the proposed scheme is higher than that of the original. More specifically, the number of stations, *N*, has to be greater than two in order for the proposed design to show better performance on queue delay. The mathematical manipulation of this comparison is shown in Equation 3.37.

$$E[\mu_{new}] > E[\mu_{old}]$$

$$\frac{B}{\tau} \frac{1+\frac{a}{2}}{N+a} > \frac{B}{\tau} \frac{1}{N} \quad \Rightarrow \quad N + \frac{a}{2} N > N + a \quad \Rightarrow \quad \frac{a}{2} N > a \quad \Rightarrow \quad N > 2 \tag{3.37}$$

The value, two, originates from the expected value of the *cwnd* ratio in Equation 3.19 of Section 3.4.1.1. If the channel condition of a queue is not as ideal, resulting in a worse *cwnd* ratio such as 1/3, this implies that the queue requires the number of queues present in the network be more than three, in order to observe better average service rate and queue delay in the proposed algorithm. In other words, the degree of improvement in performance of a queue in the new algorithm depends on the conditions of the channel. Queues with high *cwnd* ratio experience higher average service rate and shorter average delay

while bad stations can suffer. This conclusion reflects the idea of adaptively allocating the scheduling resources to desired connections, depending on the estimations of the network condition. Moreover, since the queue delay is dependent on the queue service rate, the improvement in the queue delay of the proposed scheme is also subject to bounds when an overly aggressive weight-adjusting factor coefficient is employed or when the number of queues is too large.

3.4.3 The Analysis of Round-Trip Time

This sub-section is devoted to the study of round-trip delay of a TCP segment if employing the MDRR scheme at the MAC layer. The RTT estimation at the TCP level can be split into two parts, the forward sending and the reverse returning paths, as they are expressed in the top two and bottom three lines of Equation 3.38 respectively.

$$\begin{aligned} RTT = {} & downlink\ MAC\ queue\ delay + tranmission\ time \quad (forward\ path) \\ & + propagation\ time\ in\ forward\ path \\ & + ACK\ generation\ response\ time \quad (return\ path) \\ & + uplink\ MAC\ queue\ delay + transmission\ time \\ & + propagation\ time\ in\ reverse\ path \end{aligned} \tag{3.38}$$

The transmission time of the forward and return paths can be constant if assuming the sizes of a data packet and its corresponding ACK are fixed. In addition, since the radio wave travels at the speed of light and WiMAX is a WMAN, the propagation times in the forward and reverse path are negligible in this context. The generation of a TCP ACK depends on the settings of TCP, such

as the number of accumulated ACK and the ACK delay. Assuming the settings are consistent across the queues and throughout the entire transmission session, the ACK generation response time can be considered fixed. Furthermore, if the uplink traffic is light and the size of an ACK is small, an ACK is quickly served upon arrival. Thus, the queuing delay at the uplink is negligible or consistent, comparing to the queuing delay at the downlink. With the aforementioned assumptions, the RTT estimation of a TCP segment can be rewritten as in Equation 3.39, denoting RTT_{MAC} to represent the sum of all fixed and negligible terms.

$$RTT = downlink\ MAC\ queue\ delay + RTT_{MAC} \tag{3.39}$$

Substituting the expected queuing delay from Equation 3.34, the expected RTT is found as in Equation 3.40.

$$E[RTT] = E[D] + RTT_{MAC} = \left(\frac{1}{E[\mu]} - \frac{1}{E[\lambda]} \right) \cdot F + RTT_{MAC} \tag{3.40}$$

The arrival rate at the MAC layer is equivalent to the sending rate at the TCP level. The sending rate of TCP is essentially determined by the amount of data sent in one round-trip time. More specifically, the sending rate of TCP can be simplified to the congestion window size divided by RTT as expressed in Equation 3.41. Substituting the sending rate of TCP as the arrival rate of a queue at the MAC layer, the expected value of RTT can be rewritten as in Equation 3.42.

$$\lambda(t) = \frac{cwnd}{RTT} \quad \Rightarrow \quad E(\lambda) = \frac{E[c]}{E[RTT]} \tag{3.41}$$

$$\begin{aligned} E[RTT] &= \left(\frac{1}{E[\mu]} - \frac{E[RTT]}{E[c]} \right) \cdot F + RTT_{MAC} \\ &= \left(\frac{F}{E[\mu]} + RTT_{MAC} \right) \cdot \left(1 + \frac{F}{E[c]} \right)^{-1} \end{aligned} \tag{3.42}$$

Substituting the expected queue service rate of the original and proposed design in Equation 3.23 and 3.24, the expected values of RTT are expressed in Equation 3.43 and 3.44.

$$E[RTT_{old}] = \left(\frac{\tau \cdot F}{B} \cdot N + RTT_{MAC} \right) \cdot \left(1 + \frac{F}{E[c]} \right)^{-1} \tag{3.43}$$

$$E[RTT_{new}] = \left(\frac{\tau \cdot F}{B} \cdot \frac{N + a}{1 + a/2} + RTT_{MAC} \right) \cdot \left(1 + \frac{F}{E[c]} \right)^{-1} \tag{3.44}$$

Considering the expected value of *cwnd* is identical for the original and proposed algorithm, the requirement of $E[RTT]_{new} < E[RTT]_{old}$ is illustrated in Equation 3.45, which is the same result as Equation 3.37 of the queue delay analysis.

$$\begin{aligned} &\frac{F}{E[\mu_{n,new}]} < \frac{F}{E[\mu_{n,old}]} \\ &\Rightarrow E[\mu_{n,new}] > E[\mu_{n,old}] \\ &\Rightarrow N > 2 \end{aligned} \tag{3.45}$$

Based on Equation 3.42, RTT is influenced by the queue service rate at the MAC layer. Hence, the arrival rates of queues at the MAC layer (i.e. TCP

sending rate) cannot be identical if each queue receives a differentiate service rate. Substituting the RTT expression back to TCP sending rate equation (i.e. Equation 3.41), a service-rate-dependent arrival rate is resolved as illustrated in Equation 3.46.

$$E(\lambda) = \frac{E[c]}{\left(\frac{F}{E[\mu]} + RTT_{MAC}\right) \cdot \left(1 + \frac{F}{E[c]}\right)^{-1}} = \frac{E[c] + F}{\frac{F}{E[\mu]} + RTT_{MAC}} \tag{3.46}$$

Substituting the new arrival rate as λ in the queue delay derivation in Equation 3.34 of Section 3.4.2, a new queue delay expression is formulated to reflect the queue-dependent arrival rates, as shown in Equation 3.47.

$$E[D] = \left(\frac{1}{E[\mu]} - \frac{1}{E[\lambda]}\right) \cdot F = \left(\frac{1}{E[\mu]} - \frac{\frac{F}{E[\mu]} + RTT_{MAC}}{E[c] + F}\right) \cdot F \tag{3.47}$$

In order to establish the condition of $E[D_{new}] < E[D_{old}]$, the expected queue service rate of the proposed design needs to be greater than the original. As a result, the condition required for the new algorithm to perform better than the original one is the same, regardless whether the arrival rate is assumed identical or queue-dependent. More specifically, the number of queues present in the network has to be more than two.

3.4.4 Analysis of TCP Sending Rate Incorporating the Service Rate of the MAC Layer

This sub-section attempts to formulate an expression for the sending rate of TCP, which incorporates the service rate at the MAC layer. The formulae of TCP sending rate utilized in this derivation are referenced from the work of Padhye *et al.* in [22]. The sending rate of TCP presented in Padhye's paper has two forms. The first form incorporates packet loss indications in TCP that are inferred by triple-duplicate ACKs exclusively. The second form includes the timeout mechanism in addition to the triple-duplicate ACKs, and the second form is split into two parts when limitations on *cwnd* size are considered.

The two forms, including the modifications noted in the comment paper [23], are introduced in Equation 3.48 to 3.50. Some of the notations are replaced by the symbols used in this thesis for consistency. The sending rate of TCP is denoted as the arrival rate, λ, of a queue at the MAC layer. The subscript, *TD*, denotes the triple-duplicate-ACKs loss indication of the first form, and the subscript, *TO*, represents the timeout loss indication of the second form. The symbol, *p*, indicates the probability that a packet is dropped, and the symbol, *b*, is the number of cumulative ACK. The symbol, T_0, represents the timeout value of TCP.

In addition, the notation of expected value is added to both λ and RTT to signify the average property of the two symbols in [22]. Furthermore, I attach the *TD* and *TO* subscripts to RTT to denote the possibility of having different expected RTT values in the two forms. In other words, I suggest that the expected RTT can vary if the sending rate of TCP is modelled differently.

$$E[\lambda_{TD}(p)] = \frac{\frac{1-p}{p} + \frac{3b-2}{3b} + \sqrt{\frac{8(1-p)}{3bp} + \left(\frac{3b-2}{3b}\right)^2}}{E[RTT_{TD}] \cdot \left(\frac{3b+8}{6} + \sqrt{\frac{2b(1-p)}{3p} + \left(\frac{3b-2}{6}\right)^2}\right)} \tag{3.48}$$

$$E[\lambda_{TO}(p)] = \frac{\frac{1-p}{p} + E[c] + \hat{Q}(E[c])\frac{1}{1-p}}{E[RTT_{TO}] \cdot \left(\frac{b}{2}E[c] + 1 + b\right) + \hat{Q}(E[c])T_0 \frac{f(p)}{1-p}} \quad \textit{for } E[c] < c_{\max} \tag{3.49}$$

$$E[\lambda_{TO}(p)] = \frac{\frac{1-p}{p} + c_{\max} + \hat{Q}(c_{\max})\frac{1}{1-p}}{E[RTT_{TO}] \cdot \left(\frac{b}{8}c_{\max} + \frac{1-p}{p \cdot c_{\max}} + \frac{b+6}{4}\right) + \hat{Q}(c_{\max})T_0 \frac{f(p)}{1-p}} \quad \textit{otherwise} \tag{3.50}$$

The expected value of the congestion window size is given in Equation 3.51. The term $\hat{Q}(\omega)$ represents the probability that a packet lost in a window of ω is a timeout event. The term $\hat{Q}(\omega)$ is presented in Equation 3.52, and the term $f(p)$ is shown in Equation 3.53. For detailed derivation of the formulae, please refer to [22].

$$E[c] = \sqrt{\frac{8(1-p)}{3bp} + \frac{(3b-2)^2}{9b^2}} - \frac{3b-2}{3b} \tag{3.51}$$

$$\hat{Q}(\omega) = \min\left(1, \frac{\left(1-(1-p)^3\right)\left(1+(1-p)^3\left(1-(1-p)^{\omega-3}\right)\right)}{1-(1-p)^{\omega}}\right) \tag{3.52}$$

$$f(p) = 1 + p + 2p^2 + 4p^3 + 8p^4 + 16p^5 + 32p^6 \tag{3.53}$$

Denoting new symbols to represent the numerator and some terms in the denominator of Equation 3.48 to 3.50, the sending rate of TCP can be rewritten to the following forms.

$$E[\lambda_{TD}(p)] = \frac{U_1(b,p)}{E[RTT_{TD}]\cdot V_1(b,p)} \tag{3.54}$$

$$E[\lambda_{TO}(p)] = \frac{U_2(b,p)}{E[RTT_{TO}]\cdot V_2(b,p)+V_3(b,p)} \quad for\ E[c] < c_{\max} \tag{3.55}$$

$$E[\lambda_{TO}(p)] = \frac{U_3(c_{\max},p)}{E[RTT_{TO}]\cdot V_4(c_{\max},b,p)+V_5(c_{\max},T_0,b,p)} \quad otherwise \tag{3.56}$$

Substituting the above sending rates of TCP above into the expected RTT equation (i.e. Equation 3.40) derived in Section 3.4.3, the expected RTT can be expressed as presented in Equation 3.57 to 3.59.

$$\begin{aligned} E[RTT_{TD}] &= \left(\frac{1}{E[\mu]} - \frac{E[RTT_{TD}]\cdot V_1}{U_1}\right)F + RTT_{MAC} \\ &= \left(\frac{F}{E[\mu]} + RTT_{MAC}\right)\left(1+\frac{F\cdot V_1}{U_1}\right)^{-1} \end{aligned} \tag{3.57}$$

$$\begin{aligned} E[RTT_{TO}] &= \left(\frac{1}{E[\mu]} - \frac{E[RTT_{TO}]\cdot V_2 + V_3}{U_2}\right)F + RTT_{MAC} \quad for\ E[c] < c_{\max} \\ &= \left(\frac{F}{E[\mu]} - \frac{F\cdot V_3}{U_2} + RTT_{MAC}\right)\left(\frac{U_2}{U_2 + F\cdot V_2}\right) \end{aligned} \tag{3.58}$$

$$\begin{aligned} E[RTT_{TO}] &= \left(\frac{1}{E[\mu]} - \frac{E[RTT_{TO}]\cdot V_4 + V_5}{U_3}\right)F + RTT_{MAC} \quad otherwise \\ &= \left(\frac{F}{E[\mu]} - \frac{F\cdot V_5}{U_3} + RTT_{MAC}\right)\left(\frac{U_3}{U_3 + F\cdot V_4}\right) \end{aligned} \tag{3.59}$$

The expected RTT equations above indicate that RTT is affected by the service rate at the MAC layer, which is reasonable, since the round-trip delay

should decrease if the service rate increases. Substituting the RTT terms into the sending rate of TCP, Equation 3.54 to 3.56 can be rewritten as shown in Equation 3.60 to 3.62.

$$E[\lambda_{TD}(p)] = \frac{U_1 + F \cdot V_1}{\left(\frac{F}{E[\mu]} + RTT_{MAC}\right) \cdot V_1} \tag{3.60}$$

$$E[\lambda_{TO}(p)] = \frac{U_2(U_2 + F \cdot V_2)}{\left(\frac{F}{E[\mu]} + RTT_{MAC}\right) U_2 V_2 + F \cdot V_2 V_3 (1 - U_2) \cdot U_2 V_3} \quad for\ E[c] < c_{\max} \tag{3.61}$$

$$E[\lambda_{TO}(p)] = \frac{U_3(U_3 + F \cdot V_4)}{\left(\frac{F}{E[\mu]} + RTT_{MAC}\right) U_3 V_4 + F \cdot V_4 V_5 (1 - U_3) \cdot U_3 V_5} \quad otherwise \tag{3.62}$$

The above equations indicate that the service rate at the MAC layer can affect the sending rate at TCP. More specifically, the sending rate of TCP increases if the service rate at the MAC layer is higher, which is a sensible conclusion. However, in order for the service rate at the MAC layer to contribute an effect on the sending rate of TCP, the term, $F/E[\mu]$, needs to dominate compared to RTT_{MAC} in Equation 3.60. This condition is also necessary in Equation 3.61 and 3.62, but it is subject to additional restrictions (i.e. additional terms) in order for the service rate to be an influential factor in the sending rate. In other words, the proposed scheduling scheme at the MAC layer has more effect on the sending rate of TCP in a local or metropolitan area network, where the RTT_{MAC} can be kept small.

Moreover, the service rate at the MAC layer is influential if the generation of ACKs is efficient (i.e. properly calibrated settings on the number of cumulative

ACKs and the ACK delay), so the return of an ACK is prompt, thus reducing RTT_{MAC}. Finally, the term $F/E[\mu]$ is the maximum queue delay as described in Equation 3.33. Based on the mathematical induction, the effect of the MAC service rate on the sending rate of TCP may be more noticeable for queues with large buffer sizes.

This chapter has introduced the proposed cross-layer technique, including the detailed algorithm, limitations, and designs. Analytical models of average queue service rate, and queue delay were developed to understand the behavioural dynamics of the proposed scheme. The analytical models suggest that the gain of the proposed design is more observable when *N* and *a* are large. However, the gain from *N* and *a* is bounded. Furthermore, the RTT and sending rate of TCP, incorporating the service rate at the MAC layer, were analyzed. The next chapter will discuss the implementations done in the OPNET simulated model.

CHAPTER 4: AN OVERVIEW AND MODIFICATIONS OF THE OPNET MODELS

This thesis utilized OPNET Modeler® developed by OPNET Technologies, Inc. as the simulation tool. The OPNET Modeler is a discrete event simulation engine that is capable of simulating network performance, incorporating the complete stack of network protocols. TCP is one of the supported transport protocols, and WiMAX is one of the MAC layer models that are available in OPNET Modeler. TCP and its simulated model has been long developed and commonly used, so this chapter does not intend to mention the OPNET TCP model in detail, except modifications made for the purpose of this thesis. On the other hand, WiMAX is a relatively recent technology, and is anticipated to be one of the contenders for the next generation wireless metropolitan area network. In addition, the MAC layer specifications of the WiMAX standard were the subject of core modifications, as described in this thesis. Thus, the OPNET WiMAX model will be described in more detail. Before introducing the OPNET WiMAX model, a brief overview of the hierarchical modelling concept of OPNET Modeler is presented.

4.1 A Brief Modelling Concept of OPNET Modeler

In OPNET Modeler, models are built in hierarchy, with *node models* being conceptually above a *process model*. A node is an object that appears in the topology of a simulated network, and a node model defines the architectural

modules and the attributes of a node. Modules are components that generate, consume, or process a packet, in which a *processor module* is made up of a process model underlying it. The process model specifies the behavioural and logical process of a processor module, and it is developed in the Proto-C language. The Proto-C language consists of a graphical interface of a state machine, but it retains the computational compatibility with the C and C++ language. Figure 4.1 illustrates the appearance of a client node of WiMAX, and the module structure of the node model associated with the node, in which the red ellipse circles the WiMAX processor module of the node model. The process model of the WiMAX processor module is shown in Figure 4.2.

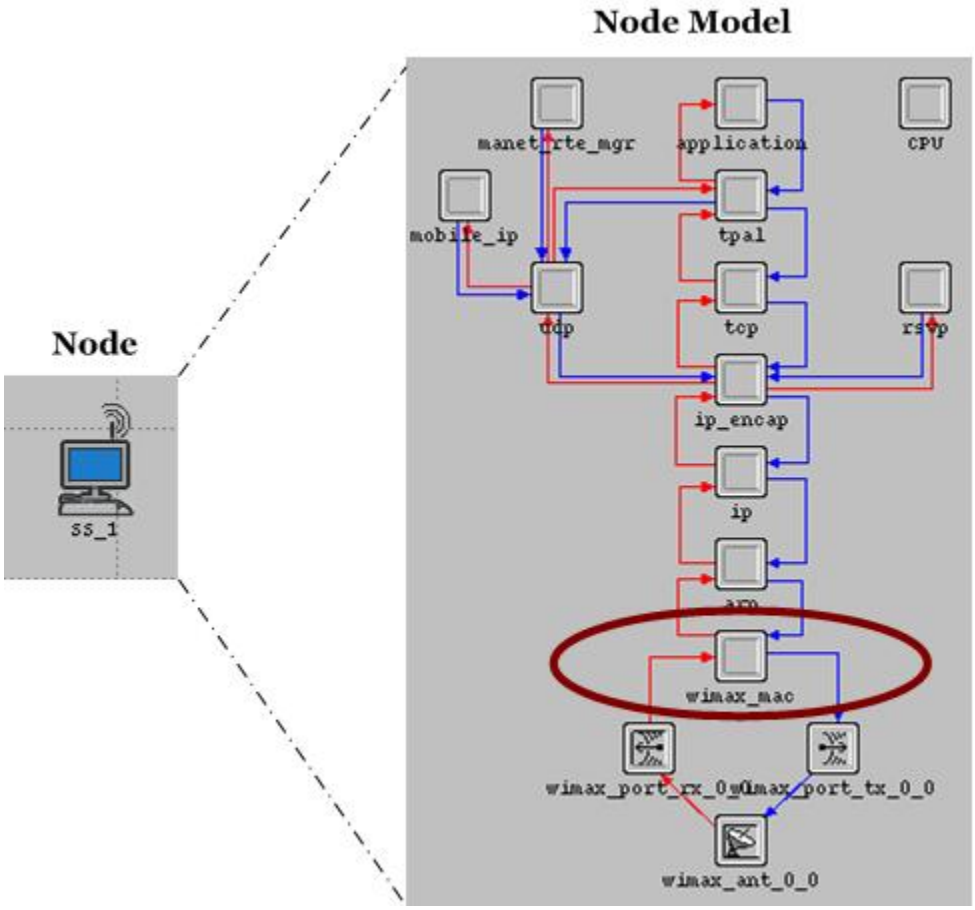

Figure 4.1: A client node of the OPNET WiMAX model and the corresponding node model

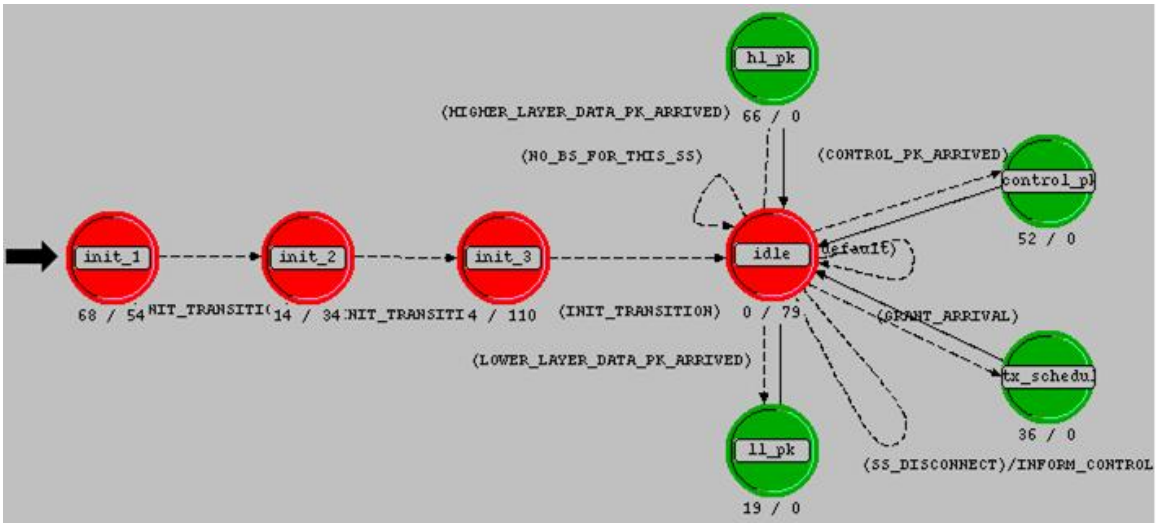

Figure 4.2: The state machine of the process model of the OPNET WiMAX processor module

4.2 The OPNET WiMAX Model in a Nutshell

The OPNET WiMAX model was under development since May 2005, and it was released in phases as more features were added. The version of the WiMAX model utilized in this thesis was the Release 5 of OPNET WiMAX model, which was available in July 2007. The model was supported in many versions of OPNET Modeler and operating systems, and the WiMAX model used in this thesis was the one that was compatible to OPNET Modeler 12.0 of the Solaris platform. The WiMAX model is still under development, and the most recent release at the time of writing is the Beta Release, which is bundled with the software of OPNET Modeler 14.0 and 14.5.

4.2.1 The Architectural Concept of the OPNET WiMAX Model

The architectural concept of WiMAX MAC is divided into two planes, where the control plane is responsible for management-related tasks, and the data plane is involved in the processing of data packets. The control plane of a

BS includes functionalities such as admission control and MAP generation. In comparison, the control plane of a SS is responsible for initial ranging of network entry and MAP decoding. On the other hand, the data plane acts as the interfaces between the WiMAX MAC layer and the adjacent layers. More specifically, the data plane of both BS and SS conveys packets across the MAC layer, and delivers the packets to the next layer. For example, one of the responsibilities of the data plane is to classify and associate an arriving MAC SDU with a CID, and generate a bandwidth request for the transmission of the SDU. The tasks performed by the data plane are common to both BS and SS. The OPNET WiMAX model is developed following the same architectural concept of one common data plane, and distinct control planes for the BS and SS.

The data plane of the OPNET WiMAX model is known as the WiMAX MAC root process model, which is composed of functionalities that are common in the data plane of a BS and SS. The WiMAX MAC root process then spawns a child process of BS or SS, depending on the role of the station, to deliver the responsibilities of the control plane. Similar to the root process, the BS and SS child processes are OPNET process models written in Proto-C language. Figure 4.3 illustrates the conceptual relationship of the root and child processes of the OPNET WiMAX model.

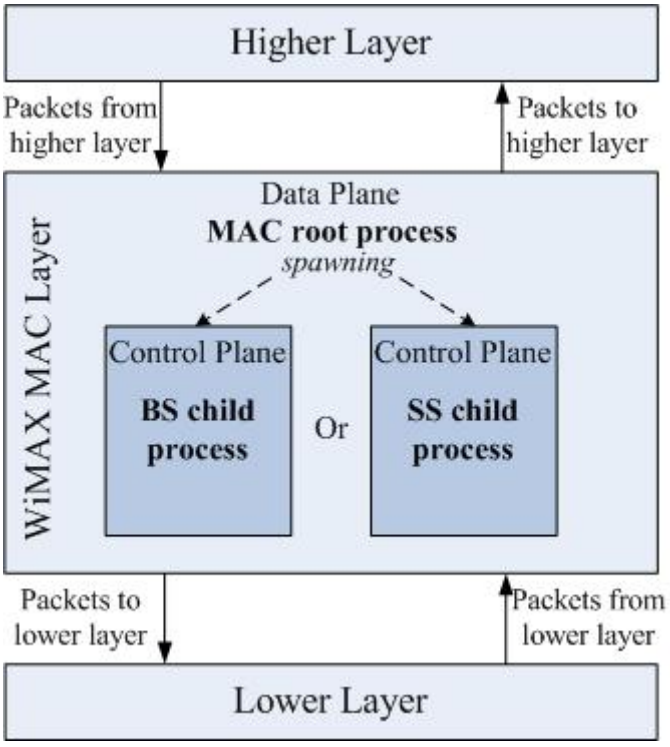

Figure 4.3: The architectural concept of the OPNET WiMAX model

While a child process is in operation, it often requires information from the root process in order to perform its functionalities. A *parent-to-child* shared memory block is established specifically for the purpose of communications between the root and child processes. The parent-to-child memory block is allocated at the creation time of the child process, and is accessible by both the root and child processes. The parent-to-child memory block stores information such as CIDs (Section 2.2.2.1), and other parameter specifications that are associated with the station. The child process retrieves necessary information from the parent-to-child memory block to accomplish its designated tasks. Upon the completions of the tasks, the child process stores the processed results in the shared memory block, and returns the control to the root process.

OPNET Modeler provides another method of communications between the root and child processes. After the initial spawning of a child process, the child process is often *invoked* by the root process when it is necessary. At each invocation, task-specific information such as the size of a bandwidth request may be passed to the child process through the use of an *argument memory*. The child process can retrieves information from either or both the parent-to-child and argument memory blocks. Unlike the parent-to-child memory block, the argument memory block is not persistent, and is created and replaced at every invocation of the child process.

4.3 Implementations in the TCP Model

This thesis involves two layers of the protocol stack, so modifications were made in both OPNET TCP and WiMAX models. As previously described in Section 3.2, a new Cwnd Option field is created in order to establish the cross-layer communication between the transport and MAC layer. Therefore, I declare a new TCP packet format to include the 32-bits Cwnd Option field in the header of a TCP segment. In addition, I introduce a new attribute in the TCP node model, of which it enables or disables the operation of the Cwnd Option field in the protocol. Upon enabling of cwnd-option attribute, TCP stores a copy of the most recent *cwnd* value to the Cwnd Option field of every TCP segment it creates. Figure 4.4 and Figure 4.5 are the screen captures of the new packet format and added attribute of the TCP module, where the modifications made are circled in red ellipses. Detailed information on the steps and coding of the implementations is attached in Appendix A.

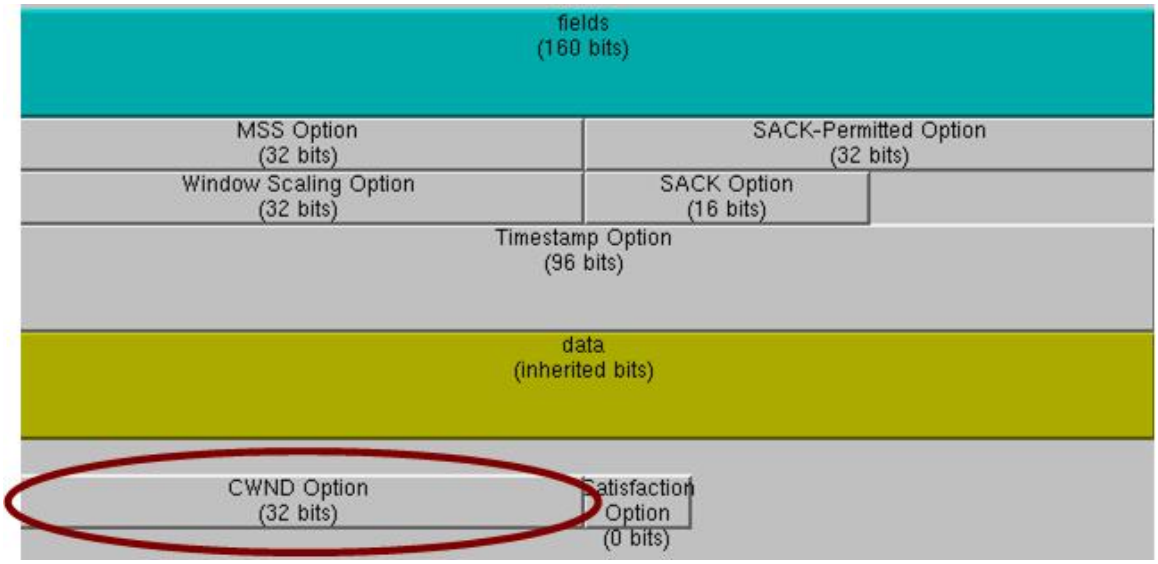

Figure 4.4: The new TCP segment format, with the modification made circled in red

Attribute	Value
RTT Gain	0.125
Deviation Gain	0.25
RTT Deviation Coefficient	4.0
Timer Granularity (sec)	0.5
Persistence Timeout (sec)	1.0
Connection Information	Print
CWND Option	Enabled
Satisfaction Option	Disabled
TCP Parameters [0].Fast Recovery	promoted
TCP Parameters [0].Maximum Seg...	promoted
TCP Parameters [0].Selective ACK...	promoted

Figure 4.5: The newly added attribute (circled in red) of the TCP module

4.4 Implementations in the WiMAX Model

As described in Section 3.3, the proposed algorithm is built on top of the MDRR queue service discipline at the downlink. Therefore, the modifications made in the OPNET WiMAX model involve only the WiMAX MAC root process and the BS-control child process.

4.4.1 Extraction and Storage of *Cwnd*

When a TCP segment traverses across the network and arrives at the MAC layer, it is *enqueued* in the buffer of a queue in the order of arrival. A bandwidth request (BWR) that reflects the size of the PDU required to transmit the packet is generated. At the same time, the *cwnd* value embedded in the TCP header is extracted from the SDU, and the value is stored at the tail end of a list structure containing integers. As the SDU is served by the scheduler leaving the queue, the corresponding *cwnd* entry is removed from the integer list. Therefore, the integer list structure contains a sequence of *cwnd* values extracted from the SDUs, and the order of the list corresponds to the order of SDUs that currently reside in the queue.

Since the OPNET WiMAX model is divided into the data and control plane, a packet arriving at the MAC layer is first processed by the root process. The task of extracting the *cwnd* value of a packet lies within the root process, and the *cwnd* value is passed to the BS-control child process, along with the bandwidth request size of the packet, in the argument memory block. When the kernel control shifts from the root to child process, the data packet itself remains in the data plane. Only necessary information is passed to the BS-control child process through memory blocks. Upon the invocation of the child process, the BWR is processed and enqueued in a queue identified by the CID in the BS control plane. At the same time, the *cwnd* value is retrieved, and stored in the dedicated list structure of *cwnd* values, in the corresponding order of the requests in the queue. Figure 4.6 depicts the concept of the *cwnd* extraction and storage procedures

implemented in the OPNET WiMAX model, where modifications are marked in red, and the code implementation is presented in Appendix B.

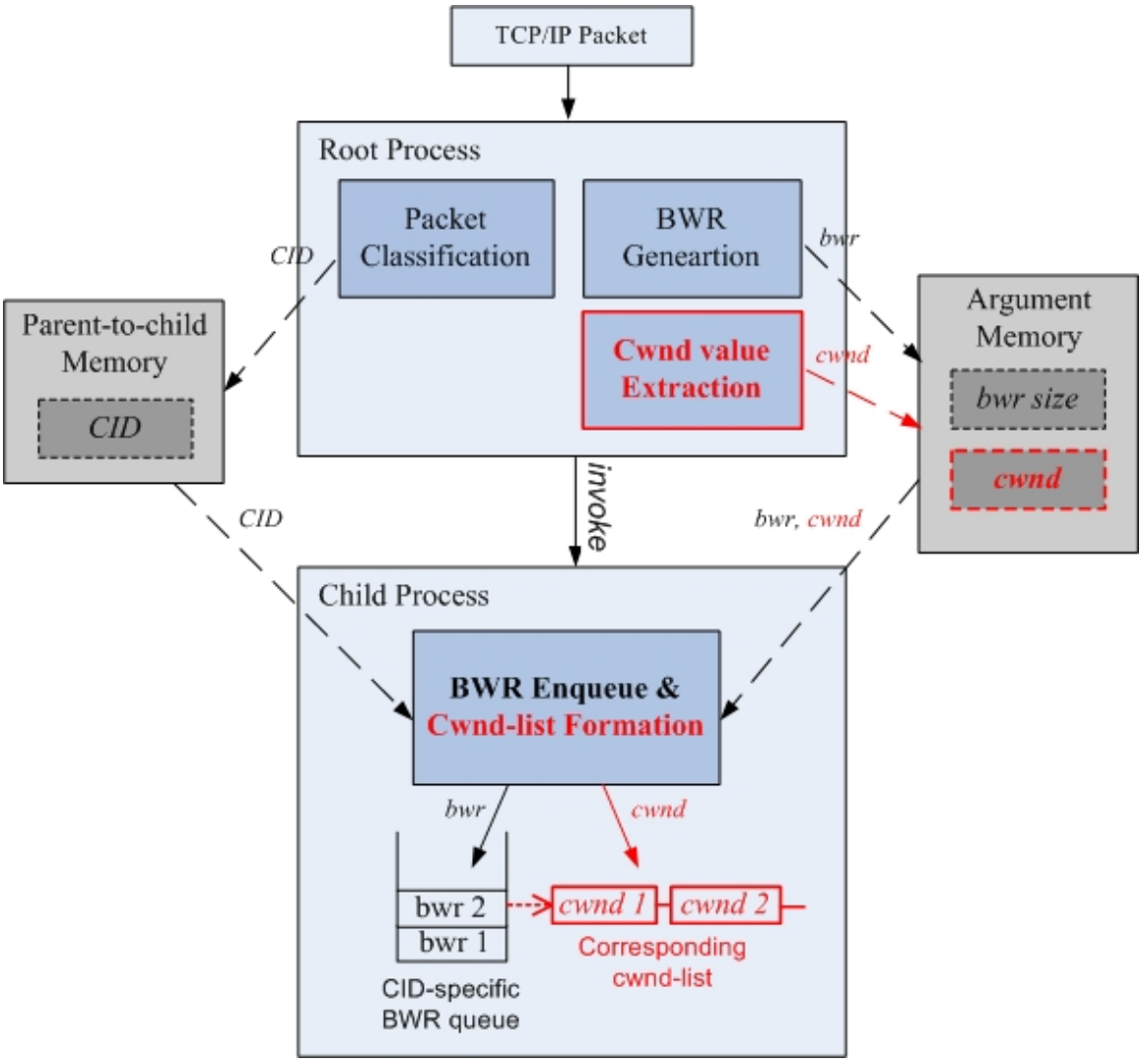

Figure 4.6: The process of the *cwnd* value extraction and storage

4.4.2 Calculation of the Queue Weight

When the BS child process is invoked again during the scheduling phase, the BS scheduler examines the BWR queues according to the MDRR queue service discipline, which has been presented in the flowchart of Section 3.3. A brief recall, the scheduler determines whether to schedule an SDU onto the next

DL subframe based on the value of the deficit counter. If the deficit counter is positive, the BS *dequeues* the corresponding request in the BWR queue, and confirms the transmission of the SDU by generating a grant. In the case when the deficit counter is zero or negative, the scheduler first replenishes the deficit counter with the weight of the queue, and examines it again. If the deficit counter recovers to positive, the scheduler initiates the dequeuing process of the packet. When a BWR is granted, the deficit counter of the associated queue is updated to its original value minus the granted size. The scheduler continues to serve the same queue if the queue is non-empty and the deficit counter remains positive. On the other hand, if the deficit counter is less than or equal to zero, the scheduler skips the queue and serves the next queue in line.

Since the weight of a queue fluctuates with the *cwnd* values in the proposed scheme, I calculate the weight of each queue before the initiation of the scheduling process. The first entry of the list structure containing the *cwnd* values of each queue is retrieved to determine the total *cwnd* values, c_t, across all queues. Then, the individual *cwnd* ratio, c_n/c_t, is calculated to resolve the cwnd-dependent queue weight, W_n'. This implementation implies that the scheduler interprets the path-wide congestion, based on the first packet of a queue. A different *cwnd* value from the list may be utilized to provide a different perspective of the network congestion condition of the flow. For example, the last entry of the list structure can be beneficial in determining the most recent congestion assessment made by TCP. Moreover, a mean value of the list structure may be useful when determining the average congestion condition of

the data flow. However, for the purpose of this thesis, only the first *cwnd* value of every list structure is extracted for the calculation of cwnd-dependent queue weight.

Though the cwnd-dependent queue weights of all queues are calculated at each round of scheduling, the weight may not necessarily be utilized to replenish the deficit counter in every round. In other words, the weight of a queue can be high or low, depending on the *cwnd* ratios, but the deficit counter only reflects the value of the weight when it is being replenished. Therefore, the logic of the scheme is regulated by the proposed algorithm but its behaviour is given a random nature. This is because the moments at which the deficit counter is being refreshed are unpredictable. This phenomenon implies that the weight of a queue is particularly important at the moments when the deficit counter is being refreshed. Nevertheless, the deficit counter is still bounded by the queue weight. Figure 4.7 depicts the conceptual process of the aforementioned scheduling process, and the queue weight calculation, with modifications marked in red. The code implementation of the algorithm is attached in Appendix C.

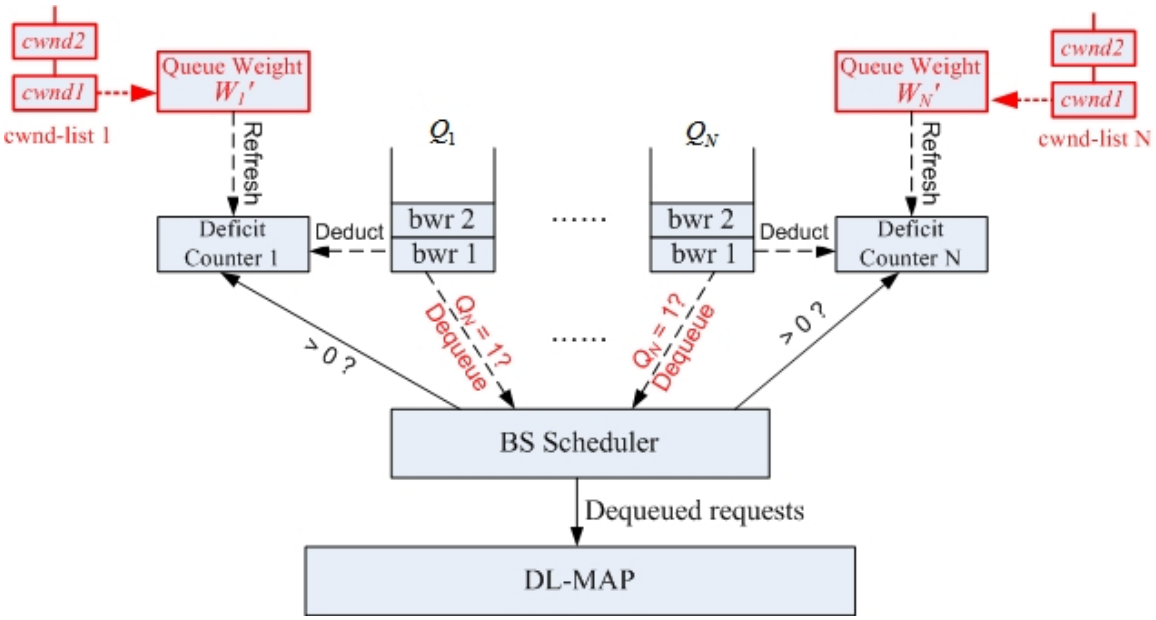

Figure 4.7: The concept of the BS scheduling process

Note that at the end of a scheduling round, the BS generates a DL-MAP, based on the dequeued requests. The DL-MAP contains data burst profiles of the granted requests, which indicates the boundaries of each data burst in the DL subframe. After the creation of a DL-MAP, the BS child process returns the control to the root process. Then, the MAC root process encapsulates the scheduled SDUs into PDUs for transmission.

4.5 Configurations of the Simulation Parameters

The topology of the simulated network is a typical last-hop wireless network, where multiple client stations (i.e. SS) are served by a centralized BS, and the BS is wired connected to a server. Figure 4.8 illustrates an example of the simulated topologies, where the number of client stations existed in the WiMAX network is two.

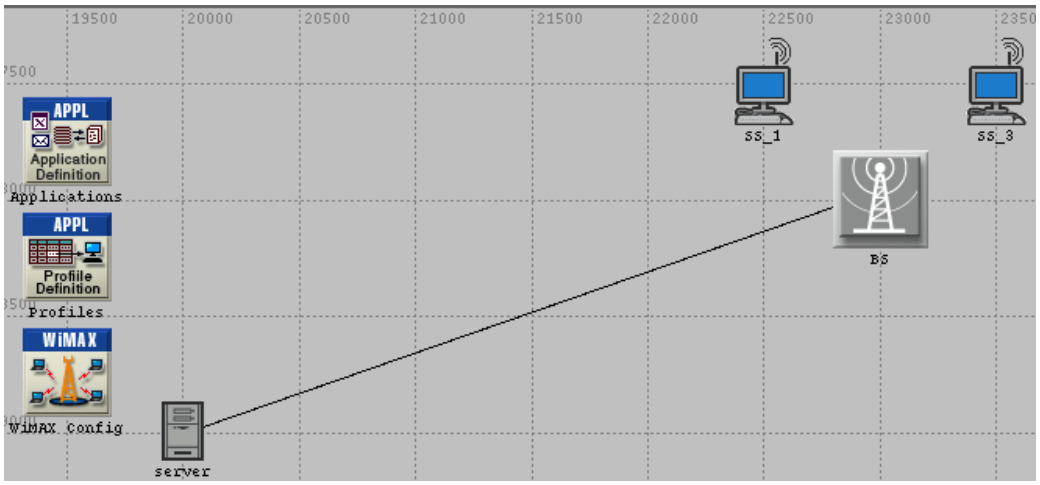

Figure 4.8: The topology of the simulated network (2 client stations Case)

The server station is wire-connected to the BS by a point-to-point duplex link that supports IP traffic, and the application specified in the server is File Transfer Protocol (FTP). The client stations are instructed to download a file size of five million bytes from the server station starting 110 seconds into the simulation time, and the same request is repeated every 50 seconds until the termination of the simulation, which is one hour. In addition, the IP ToS of the traffic is set to three, which corresponds to an *excellent* service type in IP.

4.5.1 Configurations of the TCP Parameters

At the transport layer, the TCP settings are mostly configured to the most commonly used values, except for the maximum segment size, buffer size, TCP flavour, and newly created cwnd-option attribute. At the server station, the cwnd-option attribute is enabled in order to allow TCP to write *cwnd* values to the designated option field. However, this option is not mandatory at the client stations since TCP at the client nodes only generates ACKs upon receiving of data. Thus, the *cwnd* values maintained at the receiver stations never change. In

addition, since the server station includes extra 32 bits of Cwnd Option field in the header, the maximum segment size of the proposed scheme is modified to be 32 bits less than the original. The reduction in the maximum segment size is to avoid undesirable fragmentation at the IP layer due to the newly introduced Cwnd Option field.

At the client side, the buffer size of each TCP flow is modified to 87600 bytes, which is ten times of the default setting. The purpose of this configuration is in an attempt to prevent packet loss due to buffer overflow. In other words, I intend to eliminate the constraint of buffer size on the system performance. Finally, at both the server and client sides, the TCP flavour is prompted to vary at simulation run-time. The simulated TCP flavours include Reno, New Reno, and Reno with SACK combination.

4.5.2 Configurations of the WiMAX Parameters

The scheduling service types supported in the simulated WiMAX network include rtPS, nrtPS and BE. Nevertheless, the simulation consists of only one application, FTP; therefore, only the nrtPS scheduling service type is utilized. Since each SS requests only one download in every 50 seconds, and every nrtPS connection is granted a dedicated queue, the number of queues at the MAC layer is the same as the number of client stations in the network. Furthermore, the nrtPS scheduling service type is associated with a QoS parameter, which specifies the minimum reserved traffic rate of the connection. The minimum reserved traffic rate is utilized to determine the original weight of a queue, W_n. In the simulation, the minimum reserved traffic rates of all

connections are configured to 0.5 Mbps, thus the condition of identical W_n for all N queues is established in the simulation. The uplink traffic consists of only ACK packets, but the uplink flows are also configured to be nrtPS scheduling service type for consistency.

The physical technology specified in the simulated WiMAX network is one of the OFDMA schemes. More specifically, the simulation utilized 2048 subcarriers with a corresponding channel bandwidth of 20 MHz. The modulation and coding scheme specified in the simulation is 16-QAM with 3/4 coding rate, and it is maintained the same for both downlink and uplink. Furthermore, the ARQ mechanism is not enabled for simplicity though it may be a potentially beneficial enhancement for transmissions at the MAC layer.

The physical layer condition is configured to be in free-space, which implies that the physical medium model does not incorporate multipath fading, shadowing and path loss due to signal reflections. Nevertheless, the signal is still subjected to path loss in free-space, in which the strength of a signal decays to the power of two with respect to the distance between the transmitting and receiving antennas. Moreover, the interferences and background noise are considered when determining the SNR of a signal. Based on the SNR and modulation and coding scheme of a transmitted signal, the block error rate is resolved. A block is the basic unit in a MAC frame space (Figure 2.5). The packet error rate is calculated based on the block error rate and the size of a packet in blocks. Then, a uniformly distributed random variable is compared to the packet error rate to determine whether a packet should be dropped during the wireless

transmission. Therefore, despite the fair channel quality at the PHY layer, packets are still subject to random drop in the simulation.

4.6 Validity Check of the Implemented Model

Before presenting the simulation results, the implemented model is verified against a few tests to ensure its validity. In the proposed algorithm, the cwnd-dependent queue weight is calculated based on the *cwnd* ratio, thus the weight of a queue should show a similar trend as the *cwnd* ratio. Figure 4.9 illustrates the plots of the *cwnd* ratio and queue weight, which are collected from 1000s to 1300s of the simulation time of a particular simulation run.

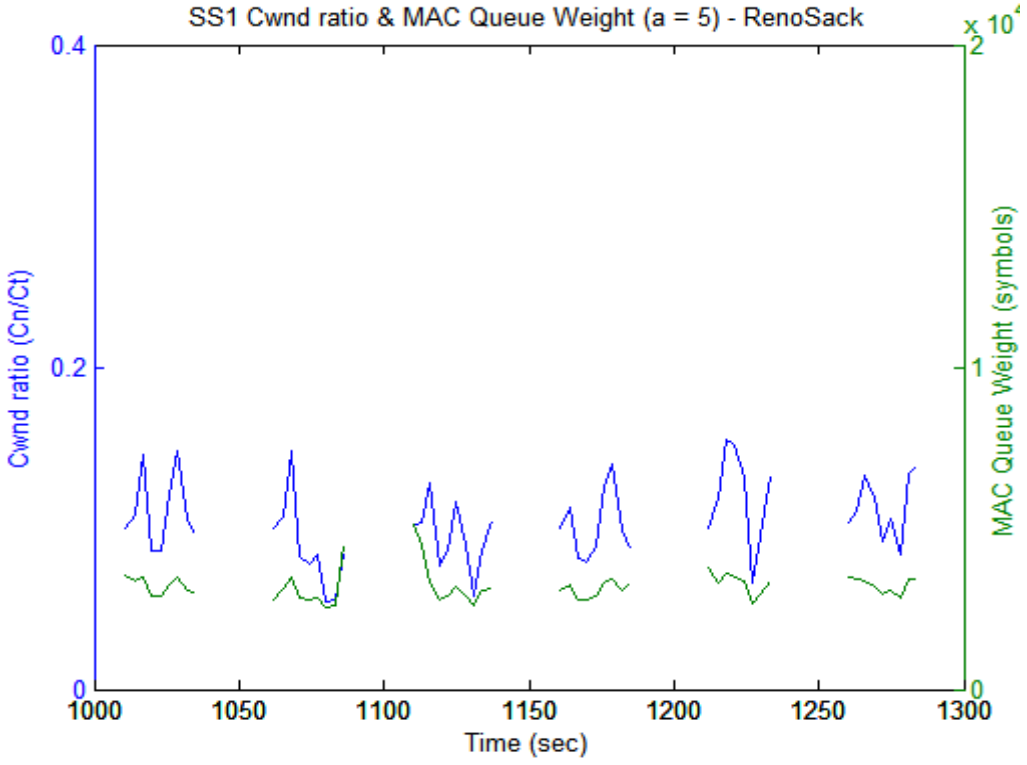

Figure 4.9: The comparison between the *cwnd* ratio and queue weight

One of the reasons for the vertical gap between the green and blue lines is the unit difference between the *cwnd* ratio and queue weight; one is unit-free, and the other one is in symbols. The unit, symbol, incorporates the modulation and coding schemes that is utilized when transmitting a packet. In other words, packets of the same size in bytes may be transmitted in different numbers of symbols, depending on the modulation and coding scheme. Despite the unit difference, the original weight of a queue is derived from the minimum traffic reserve rate, instead of the *cwnd* ratio.

The discontinuities observed in the graph when moving along the time axis are due to idling of the network, when the download of a file is complete, and the next download has not been initiated. Nevertheless, the implemented queue weight follows the fluctuations in the *cwnd* ratio as desired. Furthermore, while the queue weight follows the variation of the *cwnd* ratio, it should also show a similar trend as the congestion window size, as illustrated in Figure 4.10. Note that the reasons for the vertical gaps and discontinuities in Figure 4.10 are the same as that of the graph of *cwnd* ratio and queue weight.

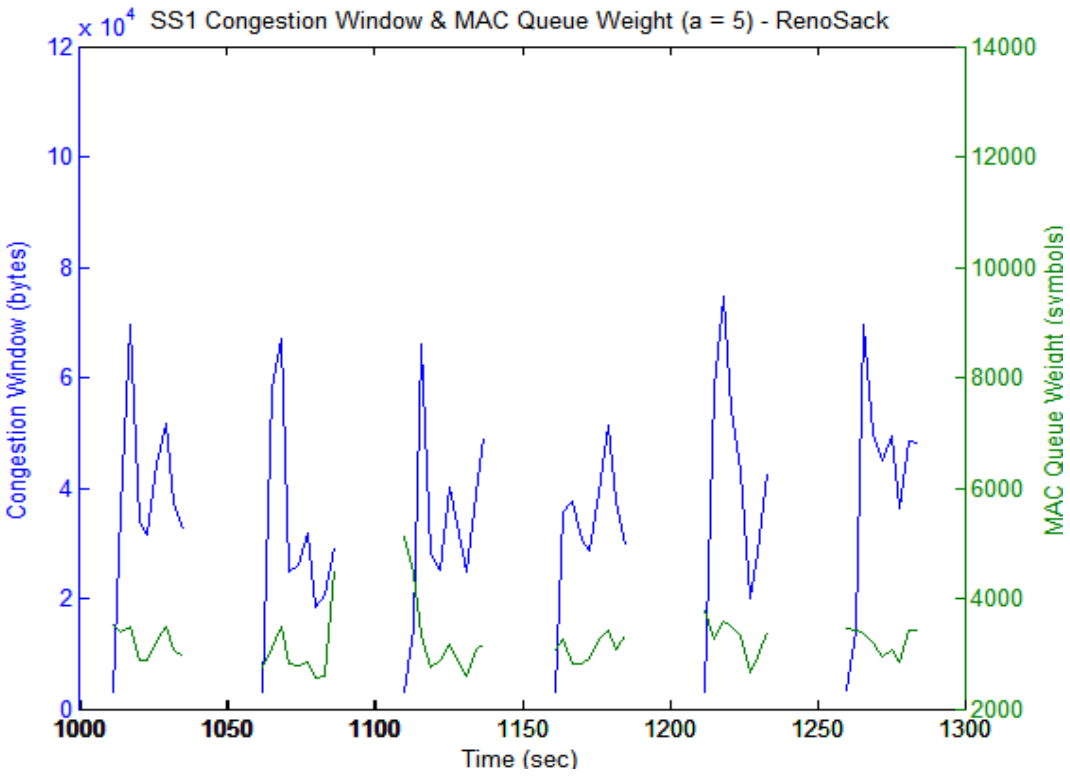

Figure 4.10: The comparison between the congestion window size and queue weight

In addition, the queue weight of each variation of the proposed design is different due to the weight-adjusting factor coefficient. Figure 4.11 demonstrates the queue weight of the same queue, but with different coefficient designs over the entire course of the simulation.

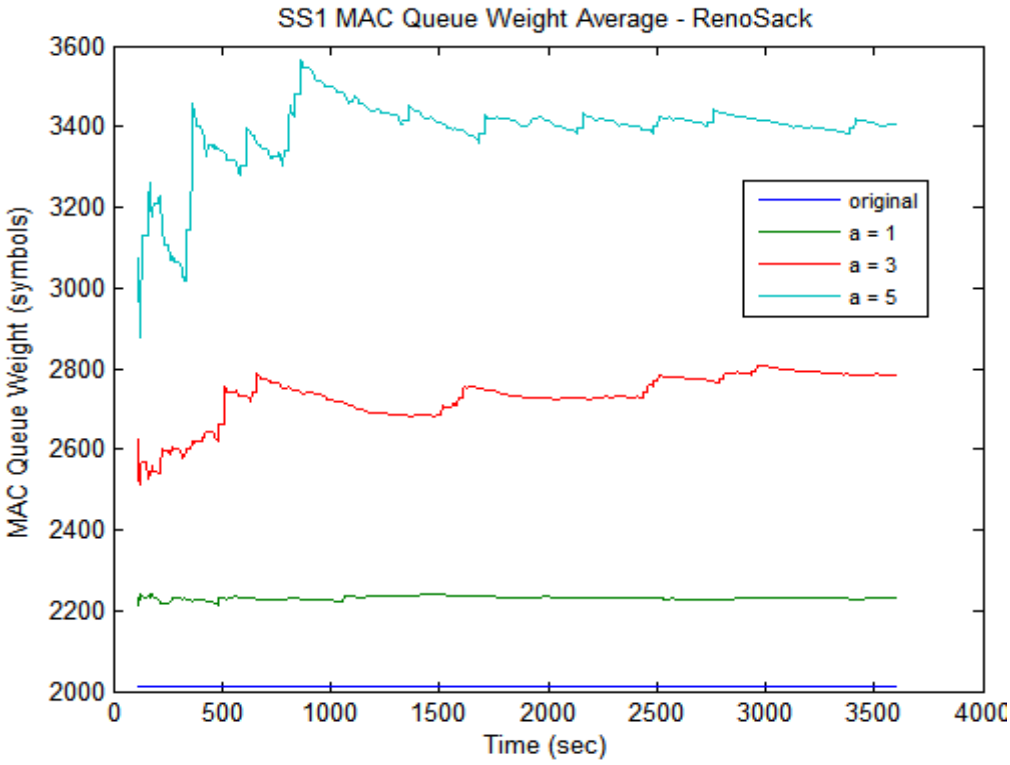

Figure 4.11: The comparison of the queue weights across different designs

The queue weight of the original design is constant, thus it appears as a horizontal line at the bottom of the graph. In comparison, the weights of the proposed designs are fluctuating, each of which varies between different ranges of y-axis depending on the value of the coefficient.

This chapter has provided an overview on the hierarchical modeling of OPNET models and the architectural concept of the OPNET WiMAX model. The implementations made on top of the OPNET models, and the configurations of the simulated network are outlined. Lastly, the validity of the implemented model is illustrated before presenting the simulation results in the next chapter.

CHAPTER 5: OPNET SIMULATION RESULTS

Traffic intensity is often an influential factor for network performance, thus simulations are conducted with respect to various numbers of stations, *N*, in the network. The values of *N* simulated in this thesis include two, four, six, eight, ten, twelve and fifteen, and each of them is simulated against four values of the weight-adjusting factor coefficient, including zero (i.e. the original design), one, three, and five. In addition, since the proposed algorithm incorporates the congestion window size of TCP, the aforementioned scenarios were simulated with various TCP flavours, Reno, New Reno, and Reno with SACK combination. The performance metrics such as delay and throughput were collected at each station.

5.1 Two Client Stations Scenario

In this section, the number of client stations presented in the WiMAX network is two, and the simulated results are organized according to the flavours of TCP. The delay and throughput statistics are collected in each scenario, and the results are introduced in the order of TCP Reno, New Reno, and Reno with SACK combination.

5.1.1 2SS – TCP Reno

The delay of a packet was determined and collected at three layers: the application, transport, and MAC layers. The delay statistics collected at the

transport and MAC layers are end-to-end delays, which were measured from the time that a TCP segment or a MAC frame was created to the time it was received by the transport or MAC layer of the receiving node. On the other hand, the delay collected at the application layer is defined as the amount of time required to complete a file download request. In other words, the download time measured the total delay of multiple packets.

The statistic results presented in this thesis are global averages of each scenario. A global average was obtained by first evaluating the mean of the data of all client stations at a given instance, and this aggregated mean of each instance was then averaged over time to obtain a global average. Thus, the resulting graph of global average is a cumulative average of the aggregated mean with respect to the simulation time. This manipulation of data points was executed on each statistic presented in this chapter, except in Section 5.7.

The global averages of the delay measurements, utilizing TCP Reno as the transport protocol, are presented in the following, where Figure 5.1 and Figure 5.2 are the end-to-end delay measured at the MAC and transport layers respectively. The download time of the requested files measured at the application layer is presented in Figure 5.3. Note that the rapid changes at the beginning of the plots are initial transient stages, where the simulations have just started and the numbers of samples are still small. The number of data points accumulates as simulations are in progress, and for stable systems, the plots enter their steady states with less fluctuation.

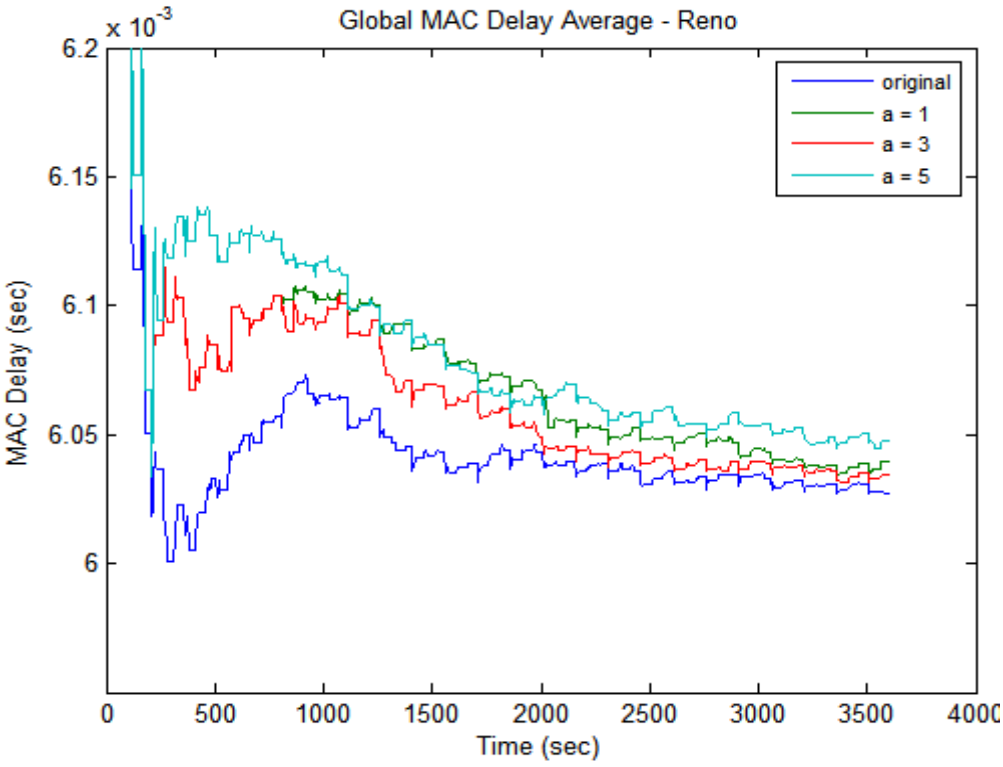

Figure 5.1: The global average of MAC delay for 2SS scenario, utilizing Reno

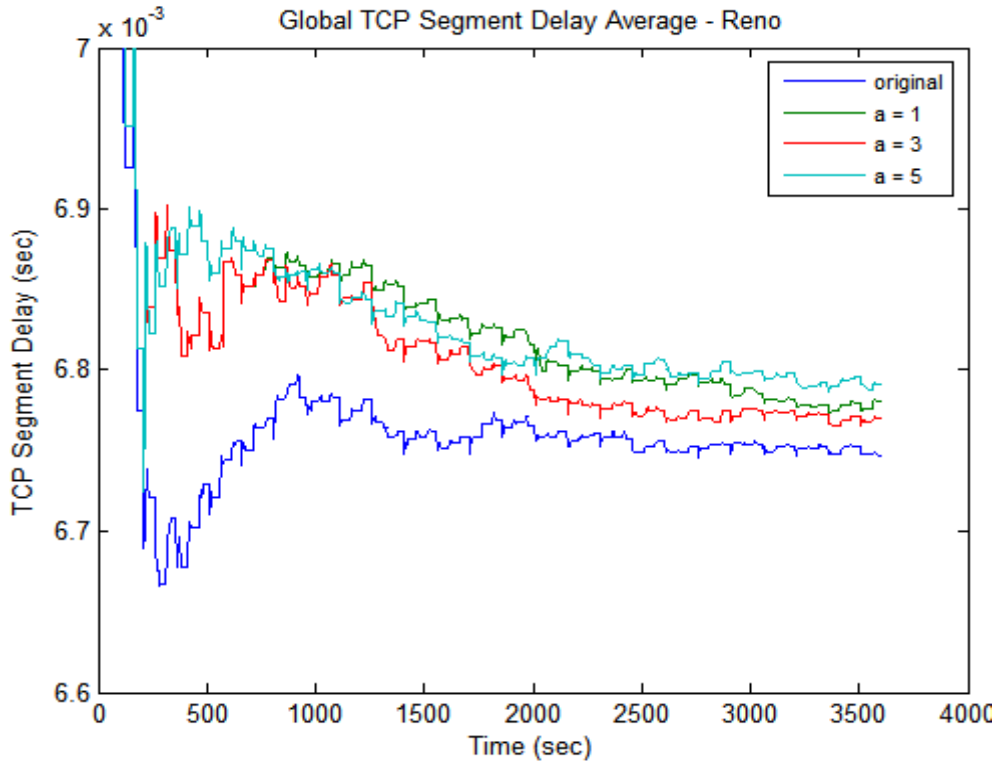

Figure 5.2: The global average of TCP delay for 2SS scenario, utilizing Reno

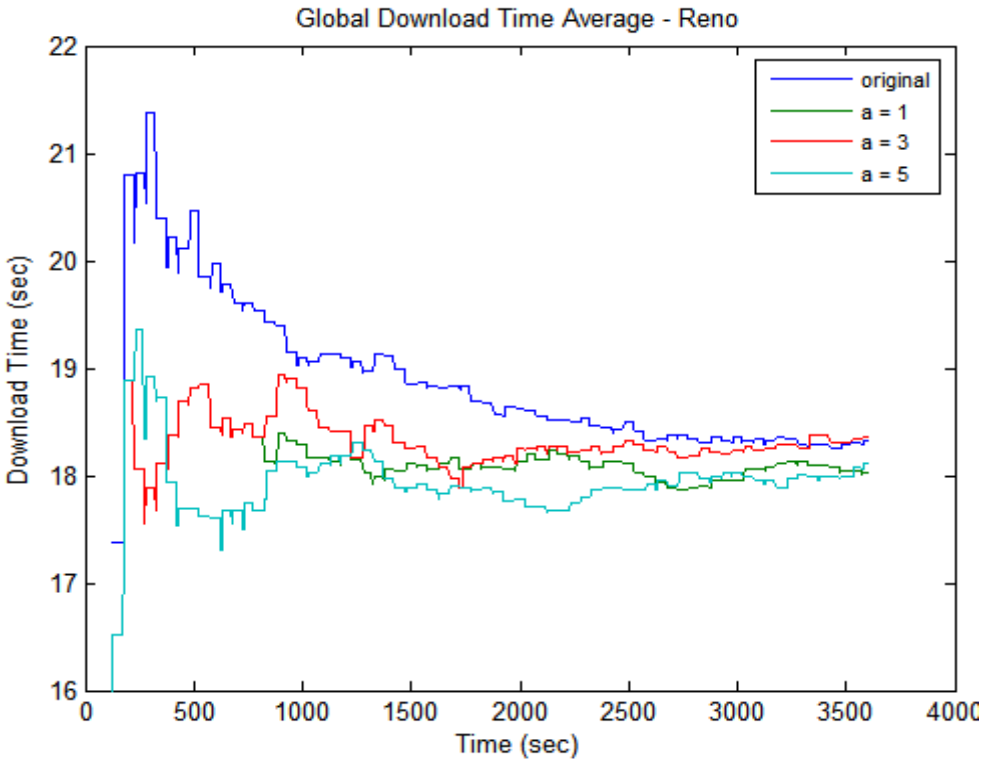

Figure 5.3: The global average of download time for 2SS scenario, utilizing Reno

The graph of MAC layer delay illustrates a close resemblance to the graph of the TCP delay over the course of the simulation time, except that the delay at TCP is higher in magnitude. TCP resides at two layers above the MAC layer; therefore, a segment is created before a frame, and a frame arrives at the MAC layer before being decapsulated into a segment and passed to the transport layer at the receiving node. Hence, the end-to-end delay measured at the transport layer is expected to be higher than at the MAC layer. However, the MAC and TCP delays of the newly proposed design are higher than the original design. The analytical model developed in Section 3.4 indicates that the number of stations, *N*, is required to be sufficiently large, two in particular, in order to offset the weight-adjusting factor coefficient, *a*, in the denominator.

Though the MAC layer and TCP delay of the proposed designs are higher than the original, the file download time of the proposed designs are better than the original. The vulnerability of TCP Reno to packet drop events, as discussed in Section 2.1.8.2, contributes a major factor to unsettle the performance at the higher layer delay. The number of packets dropped significantly affects the performance of file download time because of initiations of the timeout mechanism at TCP. As a result, the performance gain in the lower layer delays can be offset by the number of packets dropped in the PHY layer, as it is illustrated in the plots of the original and *a*=3 designs of the download time graph.

The key difference between the end-to-end delay and file download time is that the end-to-end delay measures only the delay experienced by a particular frame or segment once received. A frame or a segment that is lost in the transmission is not considered. However, the file download time incorporates the time required to recover lost packets within the downloaded file. Hence, the file download time exhibits a combined effect of the lower layer end-to-end delays and the number of packets dropped in the physical medium. The file download time is a more difficult performance metric to anticipate than the MAC or TCP delay. Figure 5.4 illustrates the number of packets dropped of each design in the PHY layer.

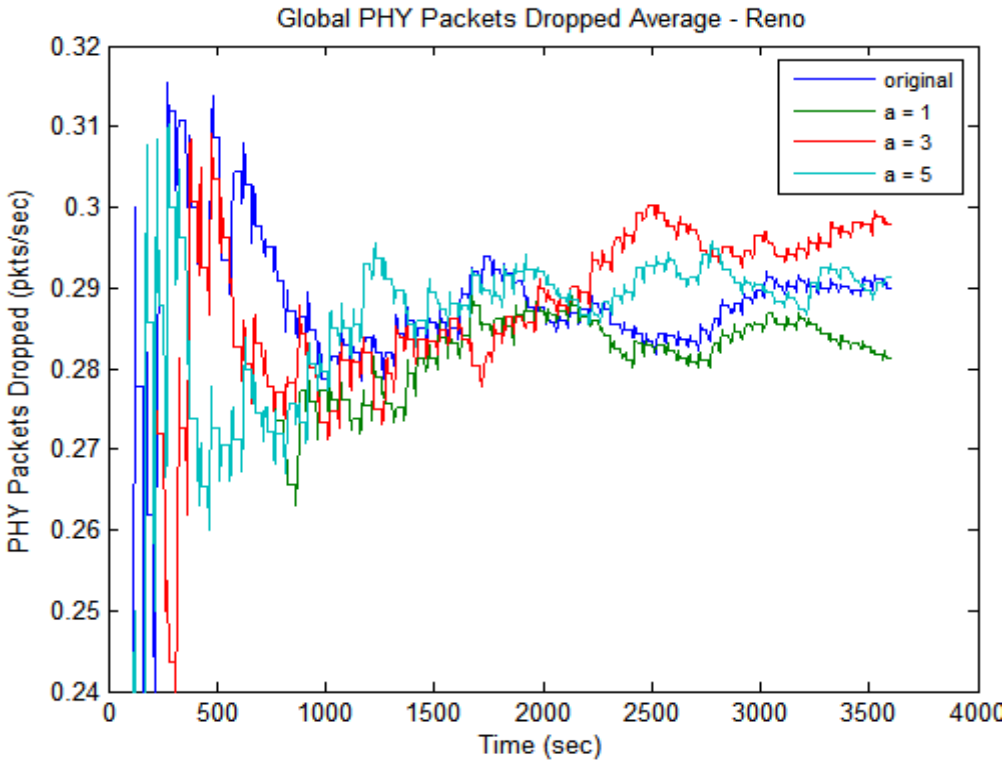

Figure 5.4: The global average of packets dropped for 2SS scenario, utilizing Reno

Note that the download time plots more closely resemble the plots of packets dropped, instead of the plots of low layer delays. This indicates that when TCP Reno is utilized and the number of stations in the network is two, the download time is significantly influenced by the condition of the PHY layer. In other words, the physical channel condition is the dominant factor in file download time when the traffic intensity of the network is low and the utilized TCP flavour is Reno.

The throughput statistic of each station is also captured, and it is measured in bits at the MAC layer, and in bytes at the transport layer. Throughput is defined as the amount of data that have been received by a node, and successfully forwarded to the higher layer. Packets that are lost during the

transmission or discarded due to error are excluded in the throughput statistic. As a result, throughput is a statistic that is affected by the physical channel quality, as is the file download time statistic. The throughput measured in this thesis is cumulative, such that the amount of data that has been successfully forwarded to the higher layer is accumulated. The accumulated value is recorded in periodic intervals, and the value is reset to zero for the next accumulation after it is recorded. Figure 5.5 and Figure 5.6 illustrates the global average of the throughput measured at the MAC and transport layer.

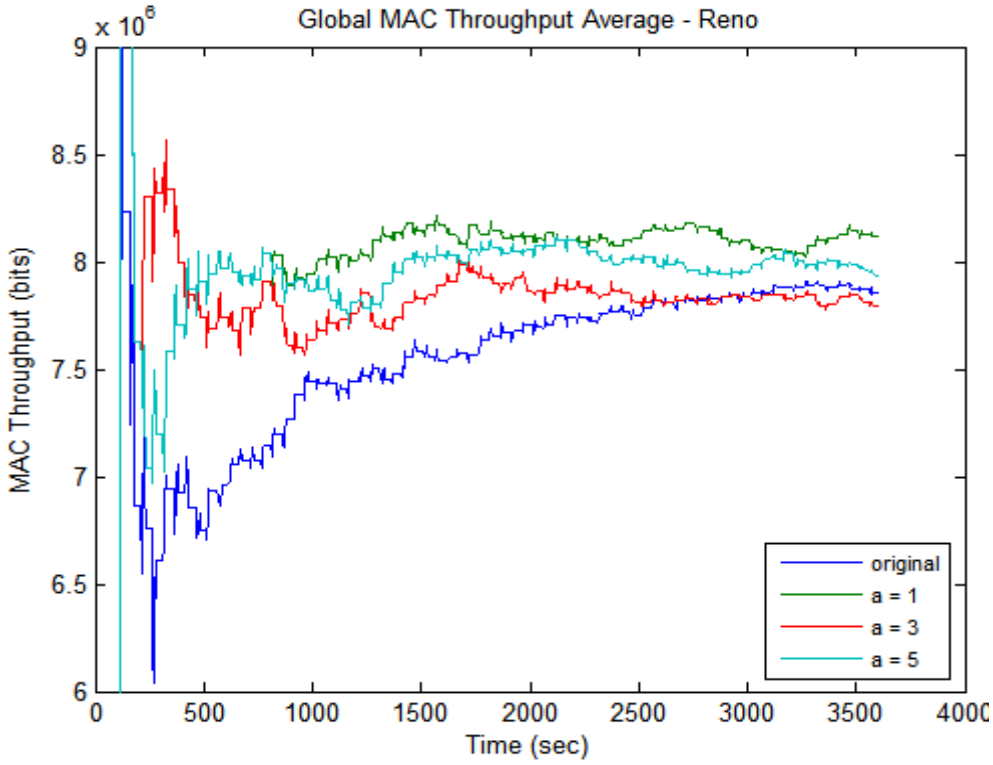

Figure 5.5: The global average of MAC throughput for 2SS scenario, utilizing Reno

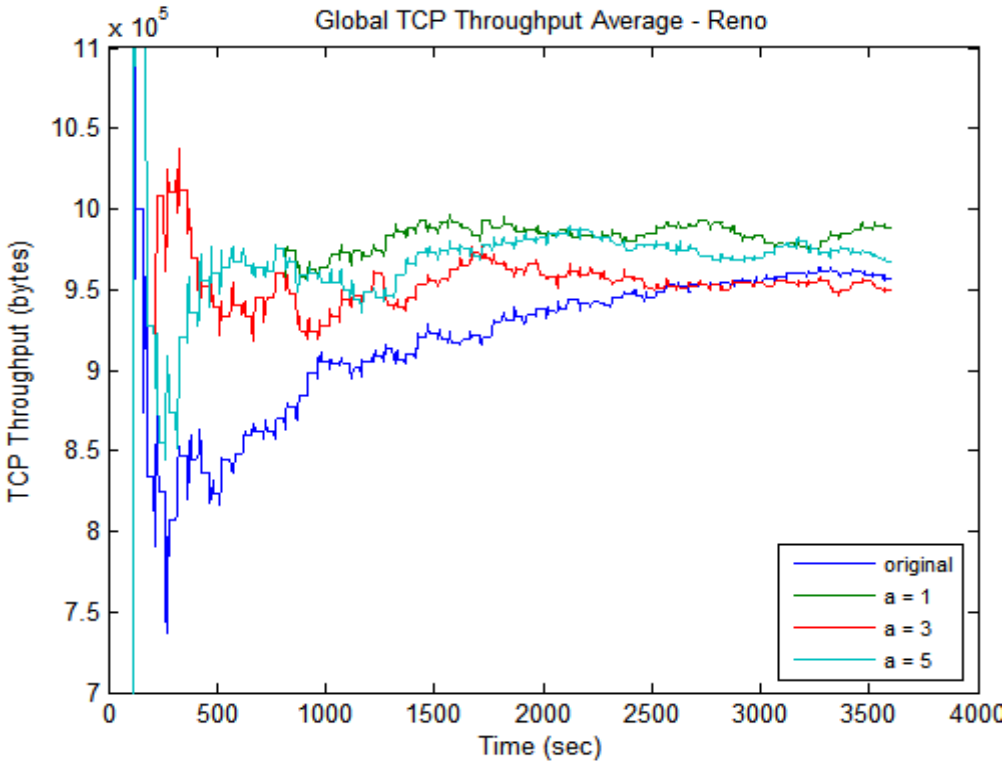

Figure 5.6: The global average of TCP throughput for 2SS scenario, utilizing Reno

The throughput measured at the MAC layer shows a close resemblance to the throughput of TCP, as was observed with the MAC and TCP delay. Nevertheless, the throughput plots demonstrate similar trends as the plots of file download time, but in the opposite direction. When throughput is low, the time required to download a file is prolonged. In contrast, the download time is reduced if throughput is high. Consequently, the throughput and file download time plots exhibit similar trends, but in an inverse direction. Though the delays of the original and *a*=3 designs are relatively low at the MAC layer, their throughputs suffer from packet drop events as observed in the graph of file download time.

5.1.2 2SS – TCP New Reno

The same simulation configurations of the TCP Reno scenario were simulated again, utilizing TCP New Reno, and the same statistics of delay, number of packets dropped, and throughput were collected. Figure 5.7, Figure 5.8, and Figure 5.9 illustrate the MAC layer delay, TCP delay, and FTP file download time respectively.

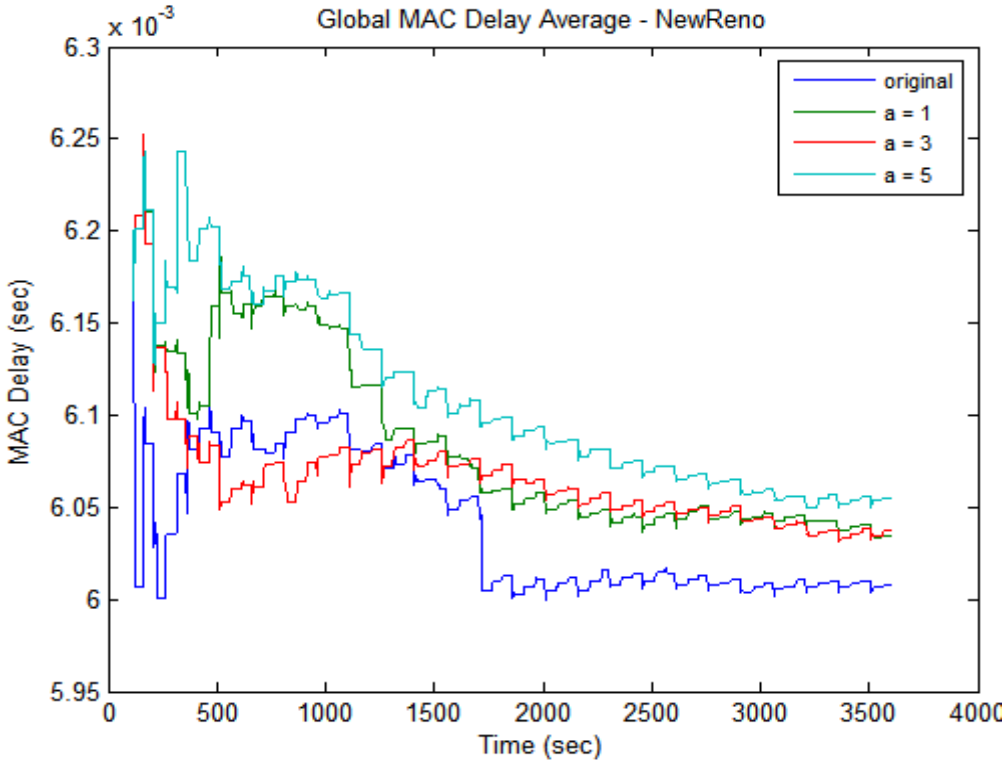

Figure 5.7: The global average of MAC delay for 2SS scenario, utilizing New Reno

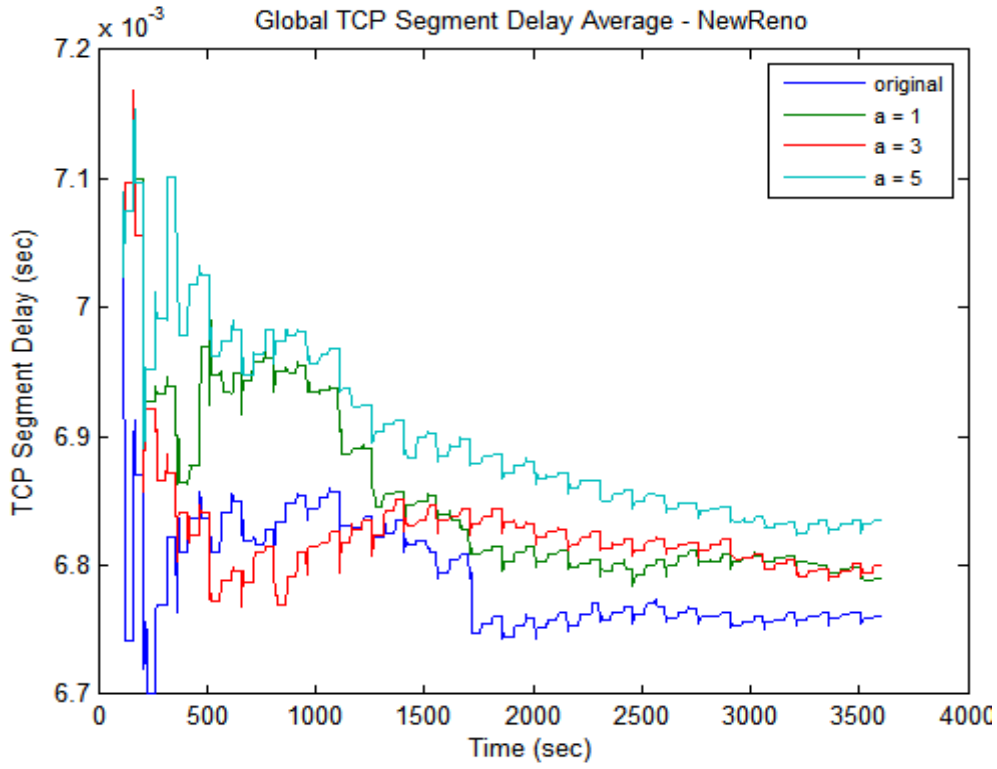

Figure 5.8: The global average of TCP delay for 2SS scenario, utilizing New Reno

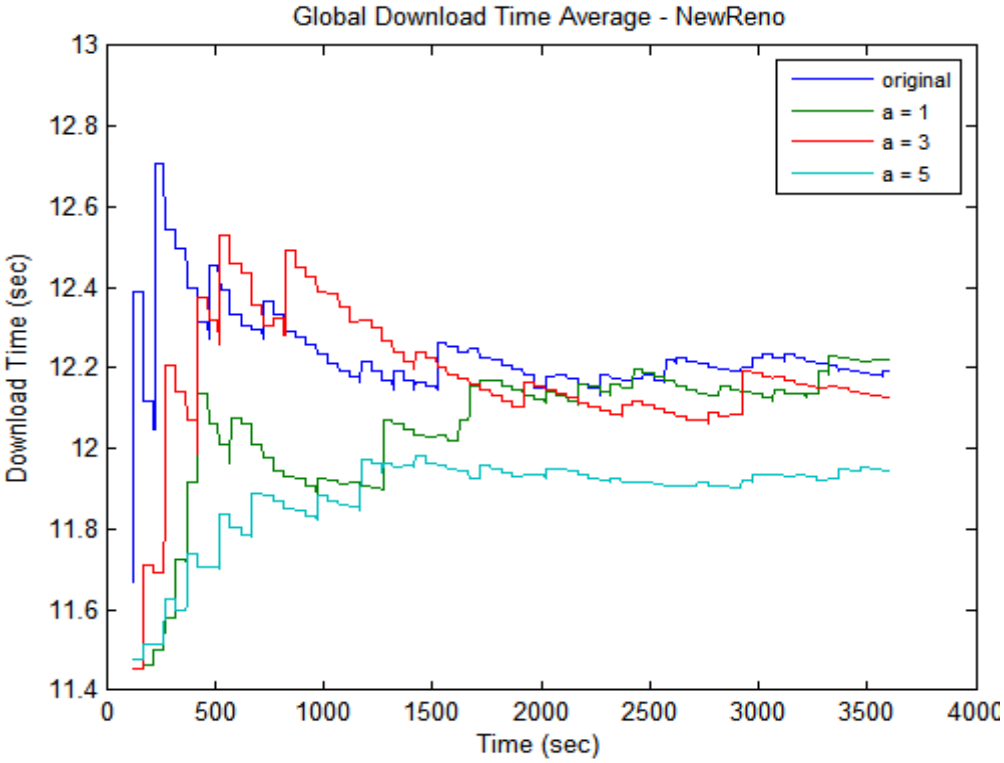

Figure 5.9: The global average of download time for 2SS scenario, utilizing New Reno

Similar observations as in the Reno scenario are noted in the New Reno simulation. The graph of the MAC layer delay closely resembles the delay graph of TCP, except the magnitude of the TCP delay is slightly higher than the MAC layer delay. Furthermore, a better end-to-end delay performance in the lower layers does not guarantee a shorter download time at the application layer. The number of packets dropped at the PHY layer is illustrated in Figure 5.10.

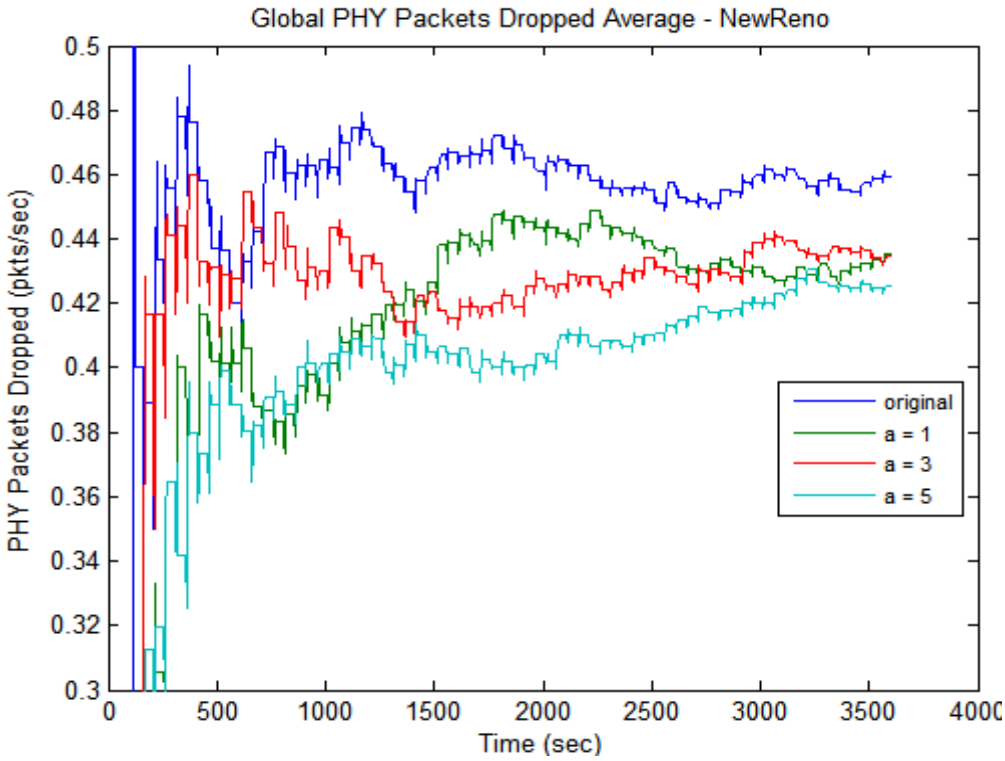

Figure 5.10: The global average of packets dropped for 2SS scenario, utilizing New Reno

Similar to Reno, the file download time is still affected by the number of packets dropped at the PHY layer. When the number of stations presented in the network is two, the file download time plots are still influenced by the number of packets dropped even though New Reno is more competent than Reno when dealing with packet drops. However, the download times collected in the New

Reno scenario are distributed at around 12-seconds range, whereas the download times in Reno are located at about 18-seconds range.

The throughput statics are also captured in the New Reno simulations. The throughputs measured at the MAC and transport layer are shown in Figure 5.11 and Figure 5.12 respectively.

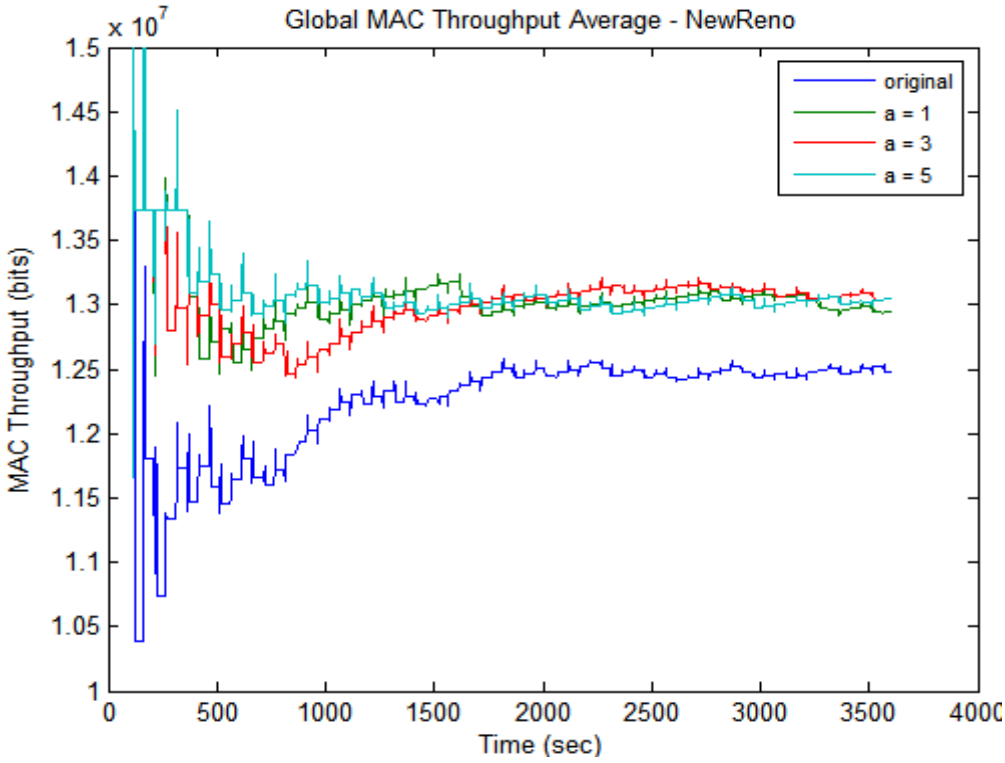

Figure 5.11: The global average of MAC throughput for 2SS scenario, utilizing New Reno

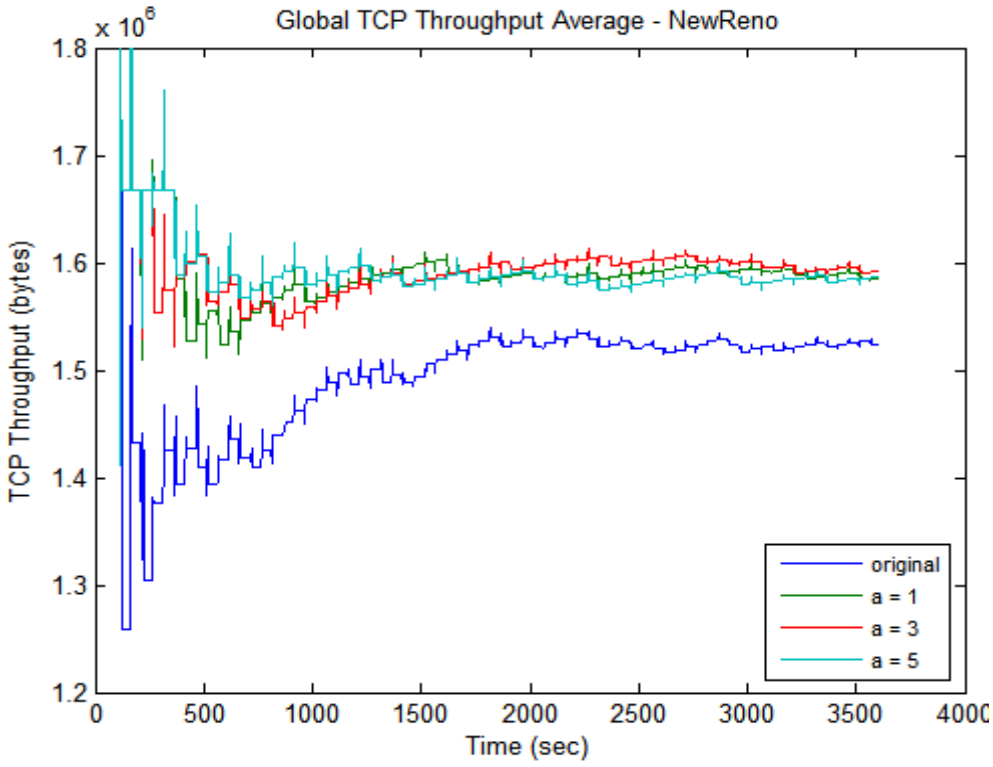

Figure 5.12: The global average of TCP throughput for 2SS scenario, utilizing New Reno

Again, the graph of throughput at the MAC layer shows a similar trend as the graph of throughput of TCP, and both statistics are affected by the PHY layer performance, as discussed in the Reno scenario. In particular, the throughput performance of the original design is reduced due to the relatively high number of packets dropped in the PHY layer. However, the average throughput in New Reno is higher than in Reno because New Reno is more adequate with packet losses. The simulation confirms that New Reno performs better in terms of throughput and file download time even if the packet drop rate is higher than Reno.

5.1.3 2SS – TCP Reno & SACK

The same parameter configurations were simulated utilizing TCP Reno with SACK combination. Since the MAC layer delay exhibits a very similar trend as the TCP delay, in the following scenarios, only the graph of MAC layer delay will be illustrated as the representation of the two. The MAC layer delay graph of the Reno-SACK combination is shown in Figure 5.13. The file download time graph is presented in Figure 5.14, and the number of packets dropped in the PHY layer is illustrated in Figure 5.15.

Similarly, the throughputs measured at the MAC and transport layers capture overlapping aspects of the network performances. Therefore, in the following scenarios, only the MAC layer throughput will be illustrated. The MAC layer throughput of the Reno-SACK combination is shown in Figure 5.16.

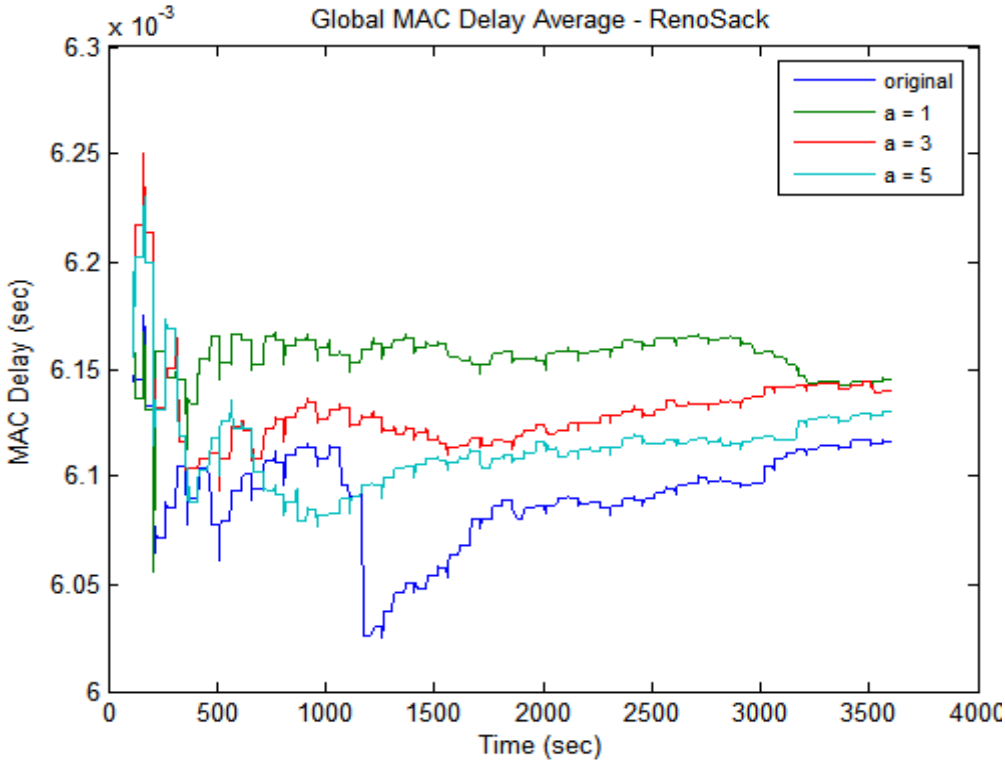

Figure 5.13: The global average of MAC delay for 2SS scenario, utilizing Reno-SACK

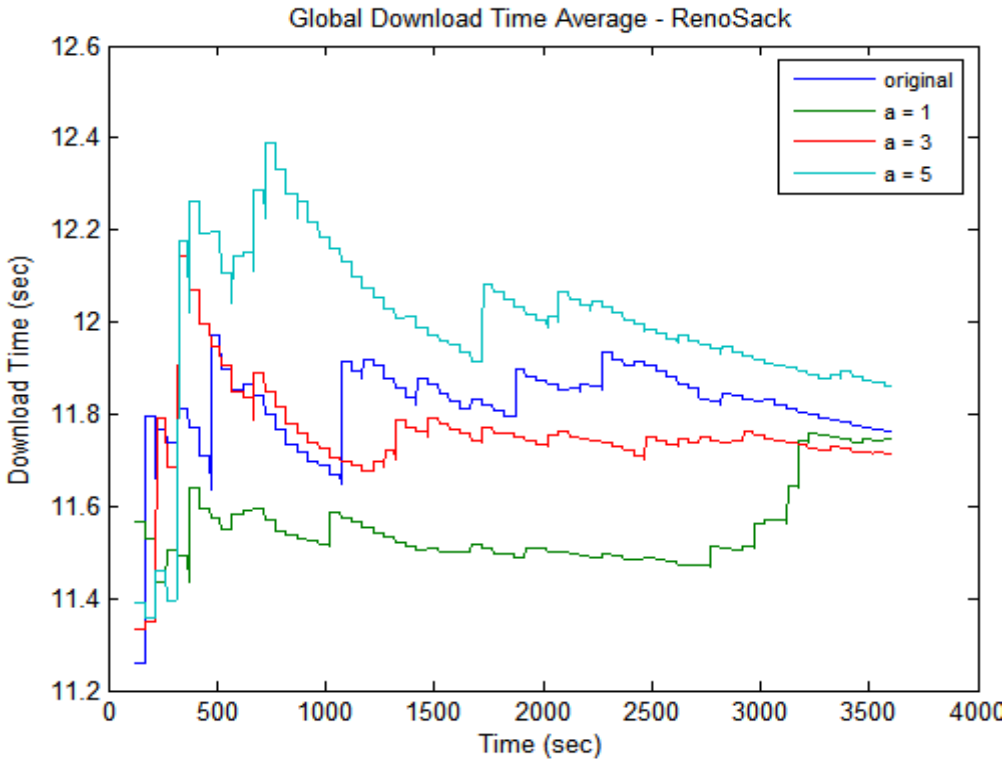

Figure 5.14: The global average of download time for 2SS scenario, utilizing Reno-SACK

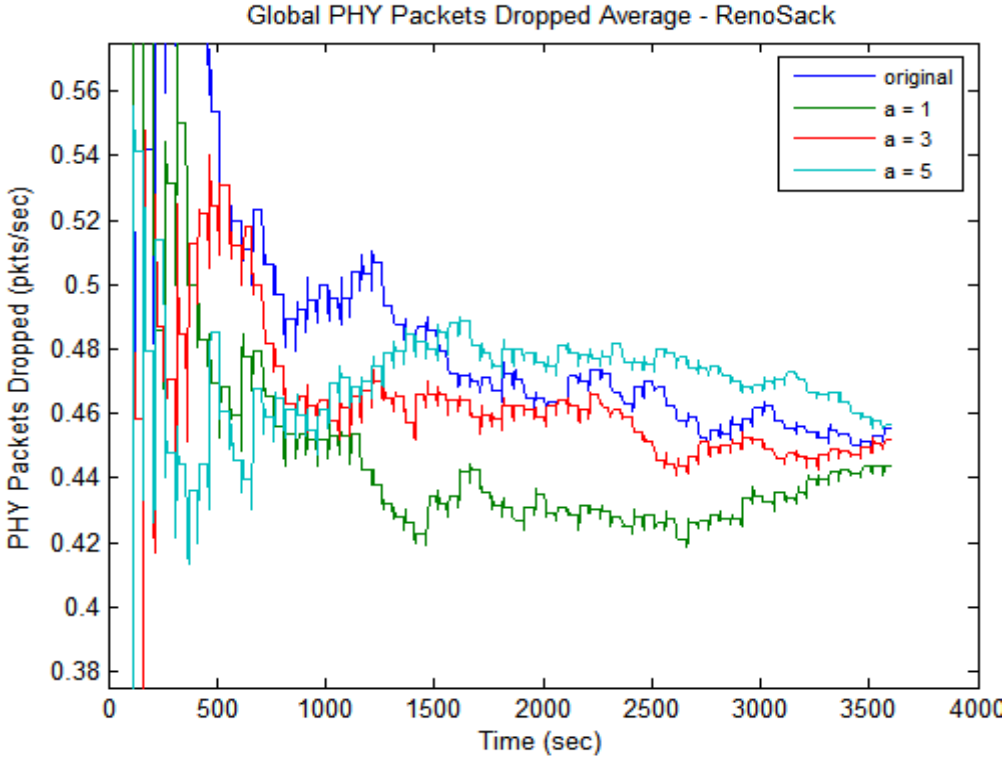

Figure 5.15: The global average of packets dropped for 2SS scenario, utilizing Reno-SACK

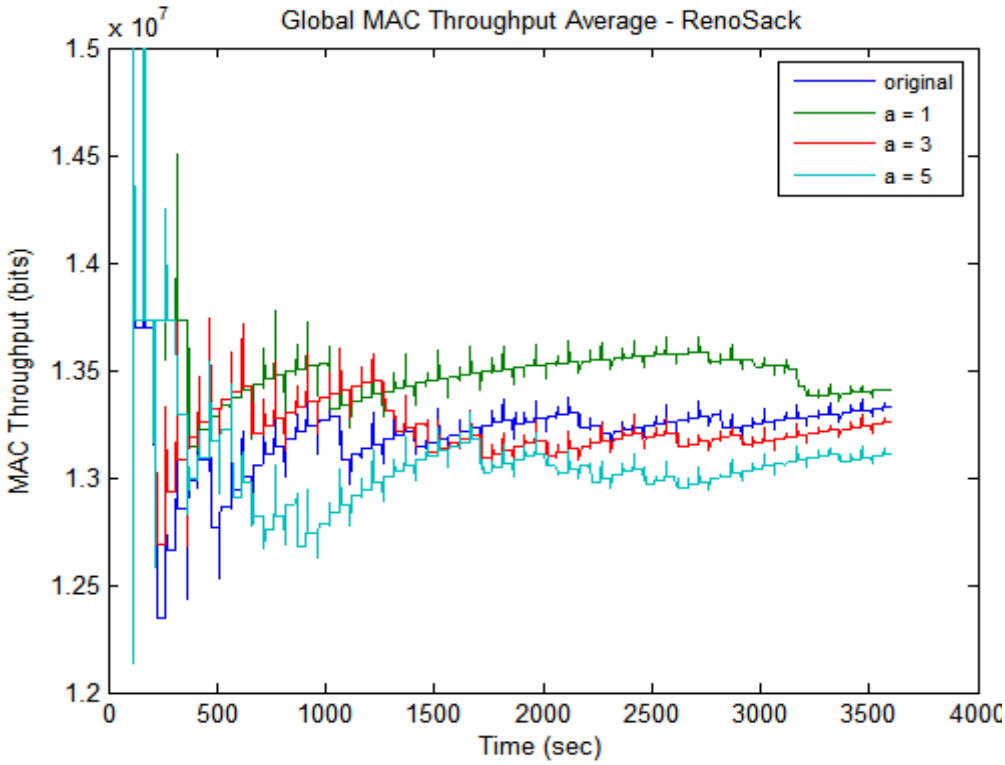

Figure 5.16: The global average of MAC throughput for 2SS scenario, utilizing Reno-SACK

For statistics that incorporate the physical channel condition such as the file download time and throughput, the performances are affected by the number of packets dropped in the PHY layer. Though the original algorithm demonstrates a better performance in the MAC layer delay, the gain is offset by the effect of packet drops at the application layer. Nevertheless, the graph of file download time in the Reno-SACK combination does not resemble as closely to the graph of number of packets dropped, as was observed in the Reno and New Reno scenarios. This implies that the physical channel condition in the Reno-SACK scenario is even less influential on the download time than in the New Reno scenario. However, the simulation results indicate that though the Reno-SACK combination is designed to elevate the resistance of TCP towards packet losses,

the performances of the file download time and throughput are still strongly affected by the number of packets dropped when the number of stations is two (i.e. the traffic is light).

The common conclusion that can be drawn from all three flavours of TCP when *N* is two is that the proposed scheme does not deliver a better performance than the original design at low layers delays (i.e. MAC and TCP delay), regardless of the values of the coefficient. This is anticipated since the benefit of the proposed scheme is expected to become apparent when *N* is sufficiently large, as analyzed in Section 3.4. In fact, if the number of stations is insufficient, the proposed algorithm performs worse than the original (Figure 3.4 and Equation 3.14), which is demonstrated in the 2SS-scenarios.

Another observation is that the performances of the file download time and throughput are dependent on the physical channel condition, thus their behaviours are more difficult to anticipate than the end-to-end delays. Therefore, the file download time and throughput are not direct indications on the effect of the proposed algorithm though they are still important statistics to consider since they are the QoS perceived by end-users.

5.2 Four Client Stations Scenario

The same simulation sequence and configuration settings are simulated in the four subscriber stations scenario. In this section and sections further on, only the MAC layer delay, file download time, number of packets dropped, and MAC layer throughput will be illustrated.

5.2.1 4SS – TCP Reno

The average of the MAC layer delay, file download time, and number of packets dropped are illustrated in Figure 5.17 to Figure 5.19.

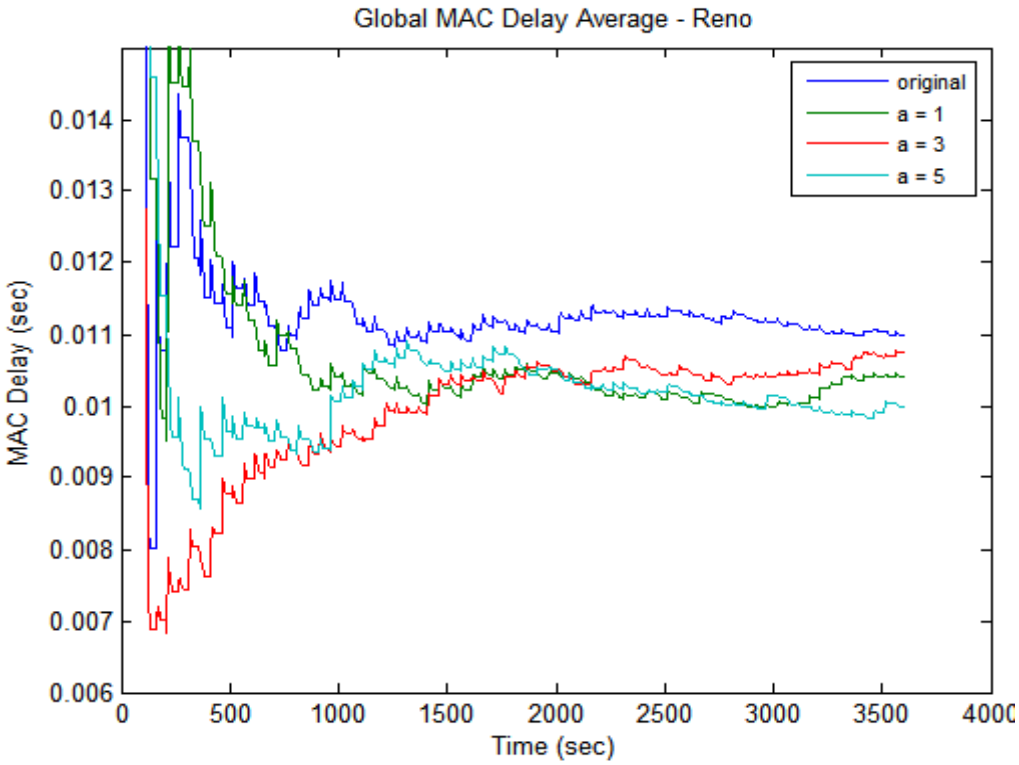

Figure 5.17: The global average of MAC delay for 4SS scenario, utilizing Reno

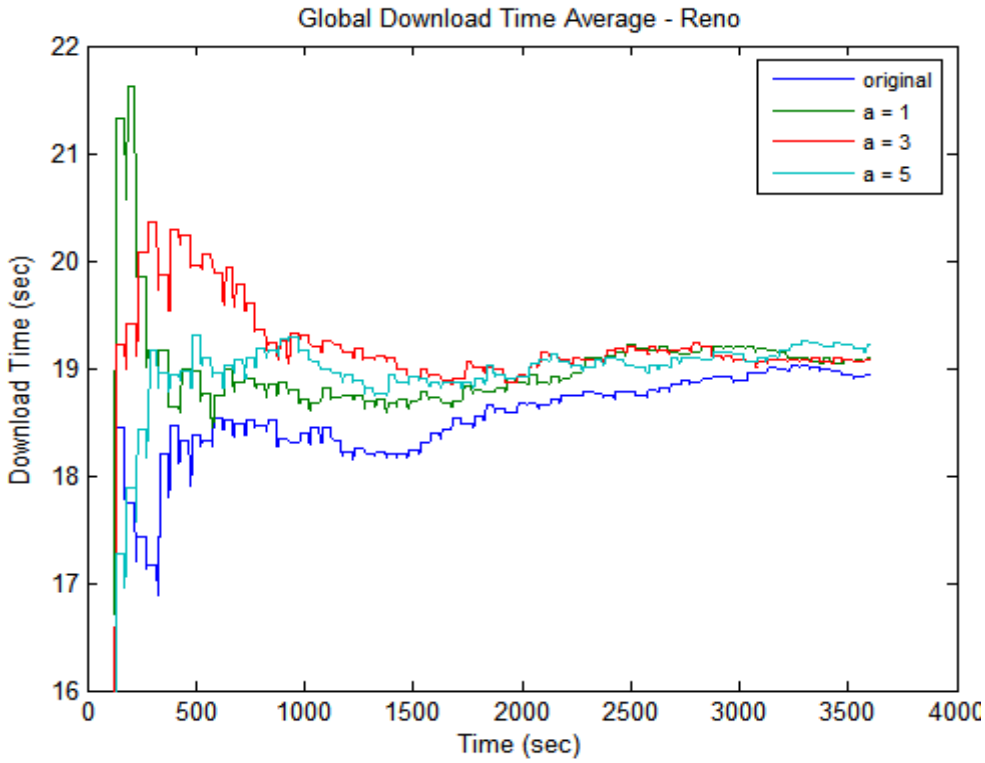

Figure 5.18: The global average of download time for 4SS scenario, utilizing Reno

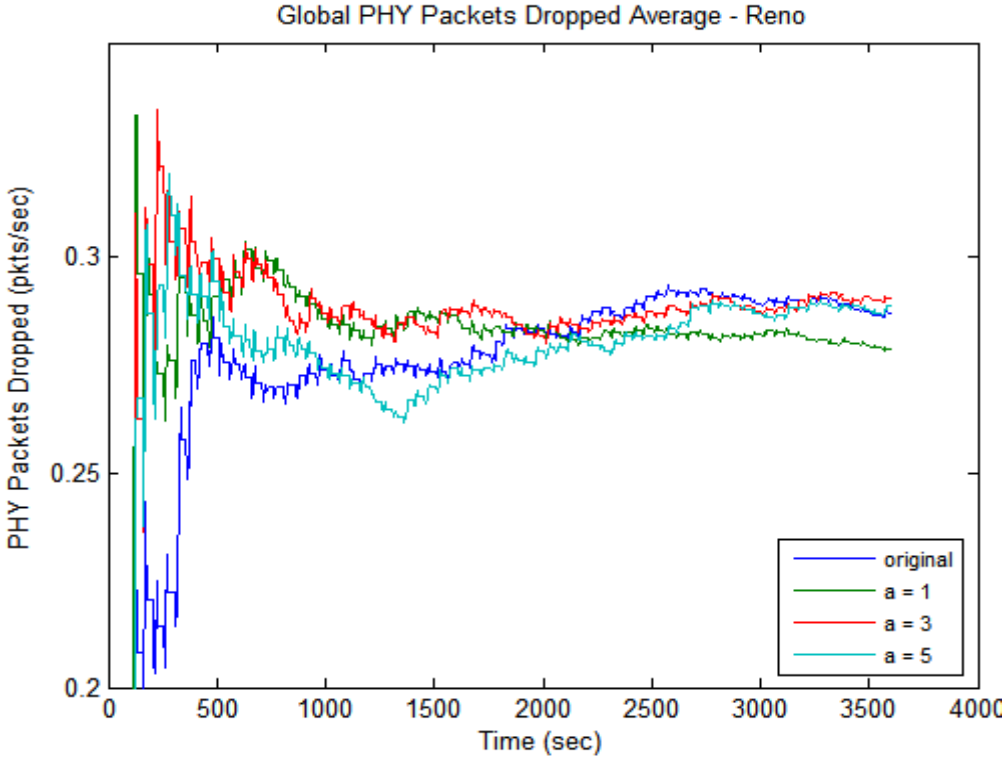

Figure 5.19: The global average of packets dropped for 4SS scenario, utilizing Reno

The MAC layer delays of the proposed designs begin to show improvements over the original design, but the file download time is again subject to changes. More specifically, the number of packets dropped at the initial stage of the original design is much lower relative to the others. Thus, the file download time at the initial stage of the original design shows a relatively small download time. As the simulation progresses, the increasing number of packets dropped in the original design results in an increasing trend in the file download time. Though the overall file download time of the original design is still relatively lower than the others, the effect of the physical channel condition on the file download time is evident. The MAC throughput of this scenario is illustrated next in Figure 5.20.

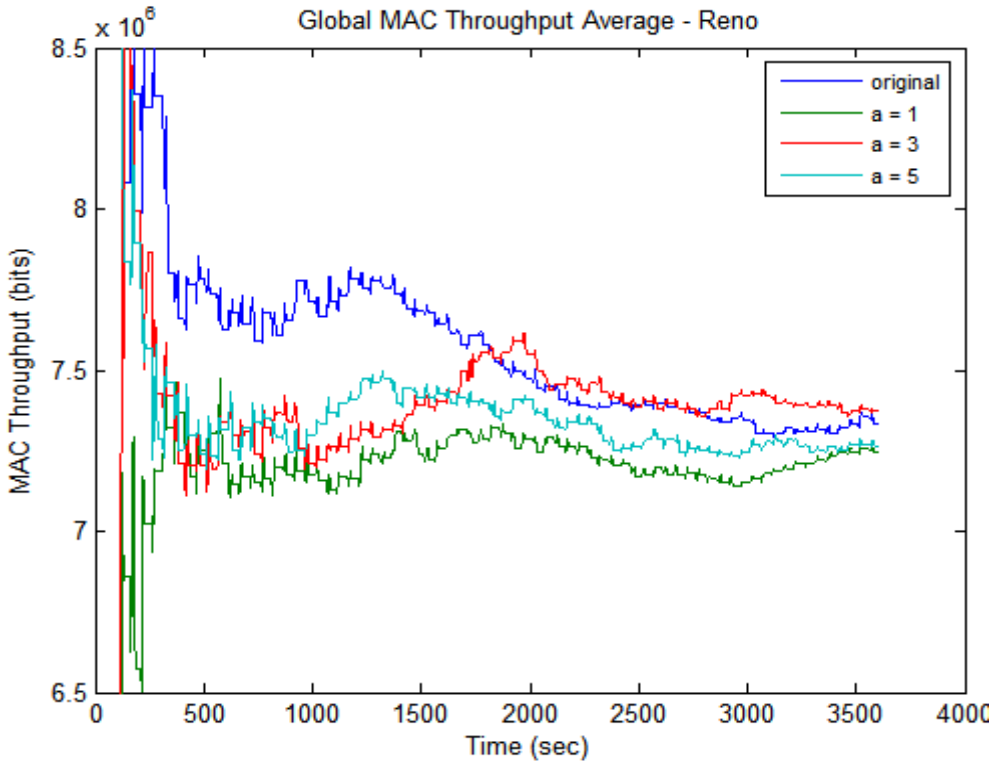

Figure 5.20: The global average of MAC throughput for 4SS scenario, utilizing Reno

The throughput plots exhibit inverse trends from the file download time plots, thus throughput is also inversely related to the number of packets dropped. More specifically, throughput of the original design is comparatively high at the initial stage, which reflects the small file download time and low number of packets dropped at the beginning of the simulation. The throughput then continues to drop as the condition of the physical channel keeps suffering.

5.2.2 4SS – TCP New Reno

This subsection presents the simulation results, utilizing TCP New Reno. The graphs of the MAC layer delay, file download time and number of packets dropped at the PHY layer are illustrated in Figure 5.21 to Figure 5.23.

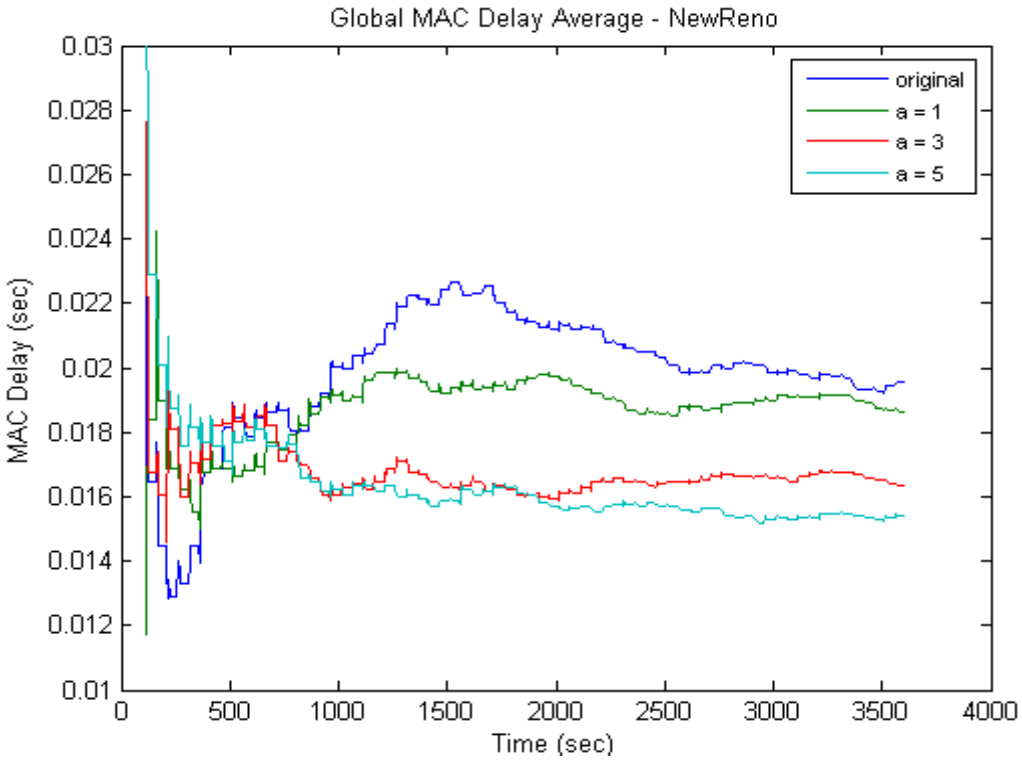

Figure 5.21: The global average of MAC delay for 4SS scenario, utilizing New Reno

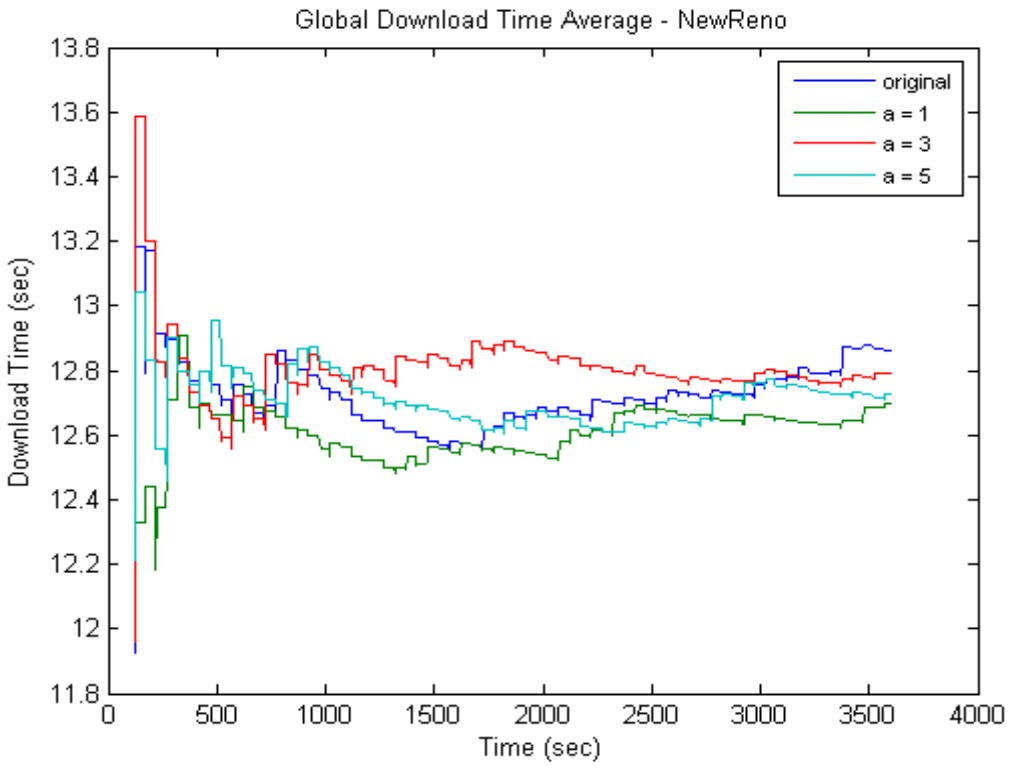

Figure 5.22: The global average of download time for 4SS scenario, utilizing New Reno

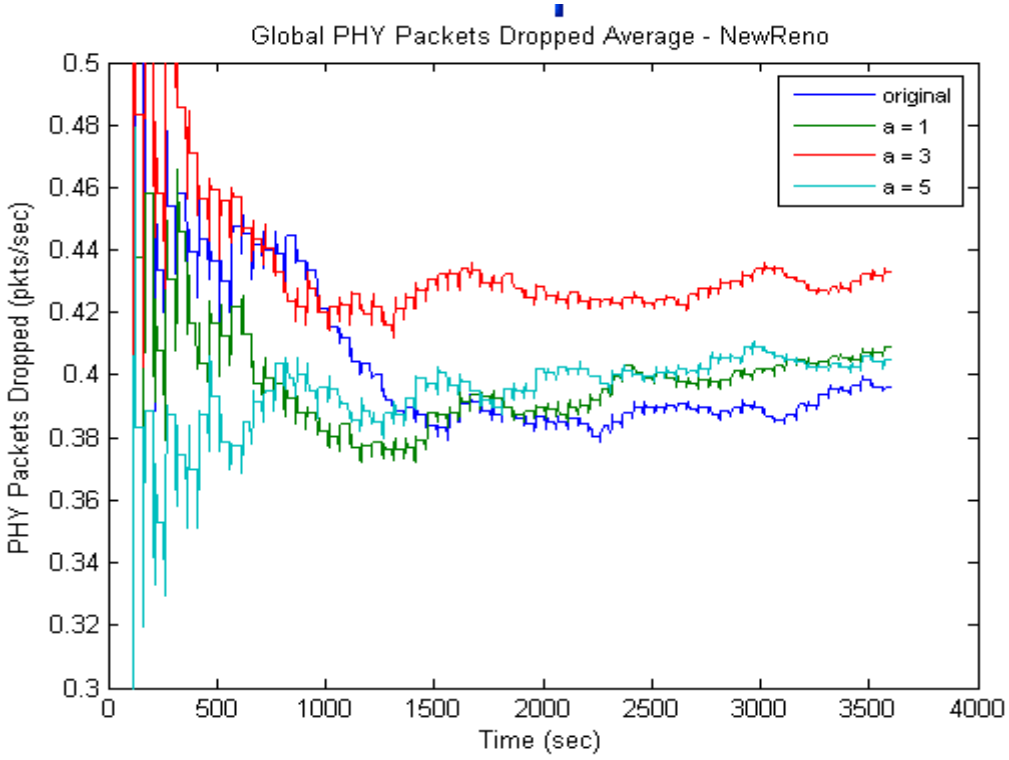

Figure 5.23: The global average of packets dropped for 4SS scenario, utilizing New Reno

In the New Reno scenario, the MAC layer delay of the proposed algorithm also shows an improvement over the original algorithm, and the improvement of each plot is more distinct than in Reno. However, the gain in the MAC layer can be compromised by significant number of packets dropped at the PHY layer. For example, the *a*=3 design shows a smaller delay at the MAC layer than *a*=1, but the high number of packets dropped causes the *a*=3 design to perform worse than *a*=1 at the application layer. Nevertheless, if the improvement at the MAC layer is significant, it can persist to the application layer. Therefore, when *N*=4, though the proposed scheme begins to show performance gain in the MAC layer delay, the improvement is not always significant enough to overcome the PHY layer condition.

Despite this, the effect of the PHY layer on the file download time is not as dominant as in the Reno or 2SS scenarios. The throughput of the MAC layer is illustrated in Figure 5.24, and the graph approximately reflects the trends in the graphs of file download time, and number of packets dropped but in the opposite direction.

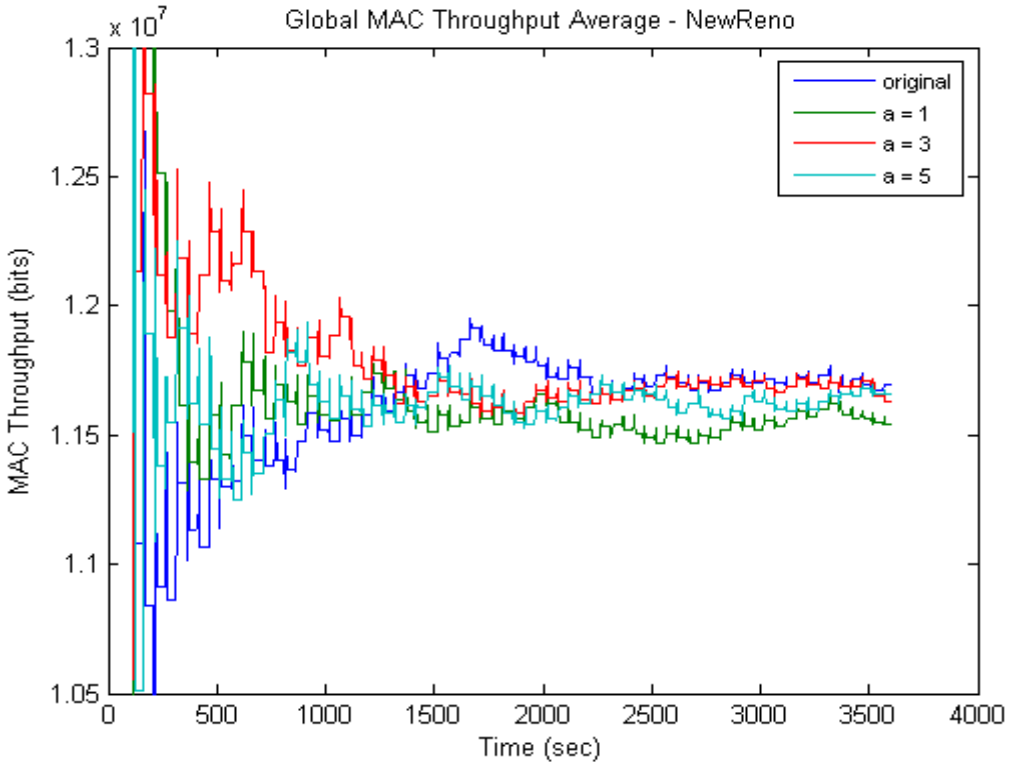

Figure 5.24: The global average of MAC throughput for 4SS scenario, utilizing New Reno

5.2.3 4SS – TCP Reno & SACK

This subsection includes the simulation results for four client stations, utilizing TCP Reno and SACK combination. The graphs of the MAC layer delay, file download time and the number of packets dropped are presented in Figure 5.25 to Figure 5.27. The graph of the MAC layer throughput is illustrated in Figure 5.28.

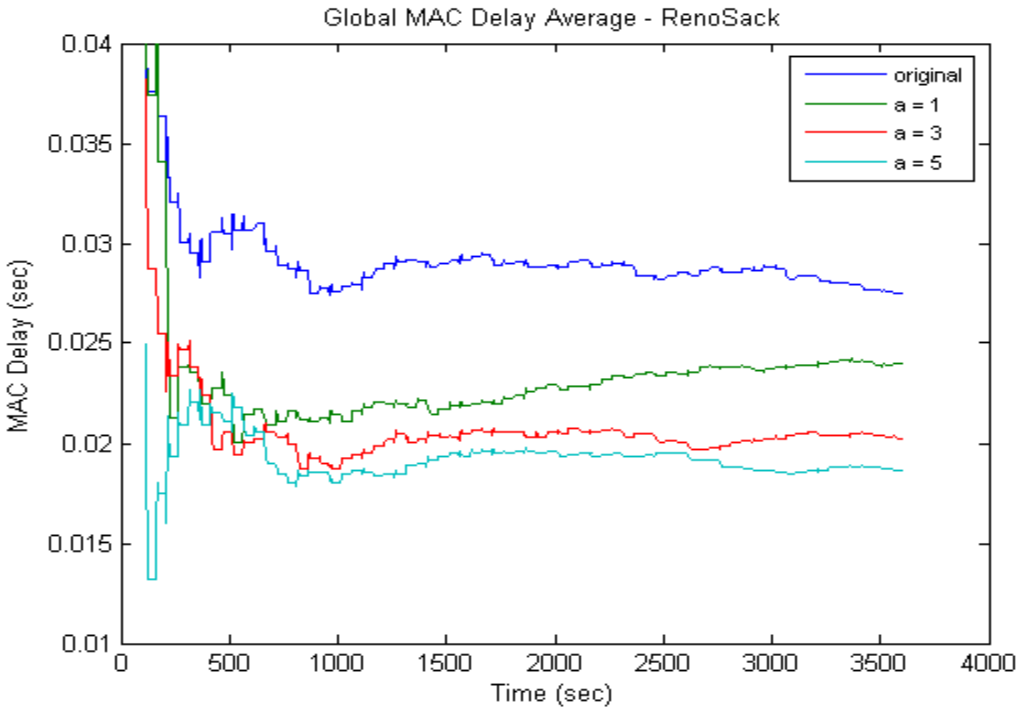

Figure 5.25: The global average of MAC delay for 4SS scenario, utilizing Reno-SACK

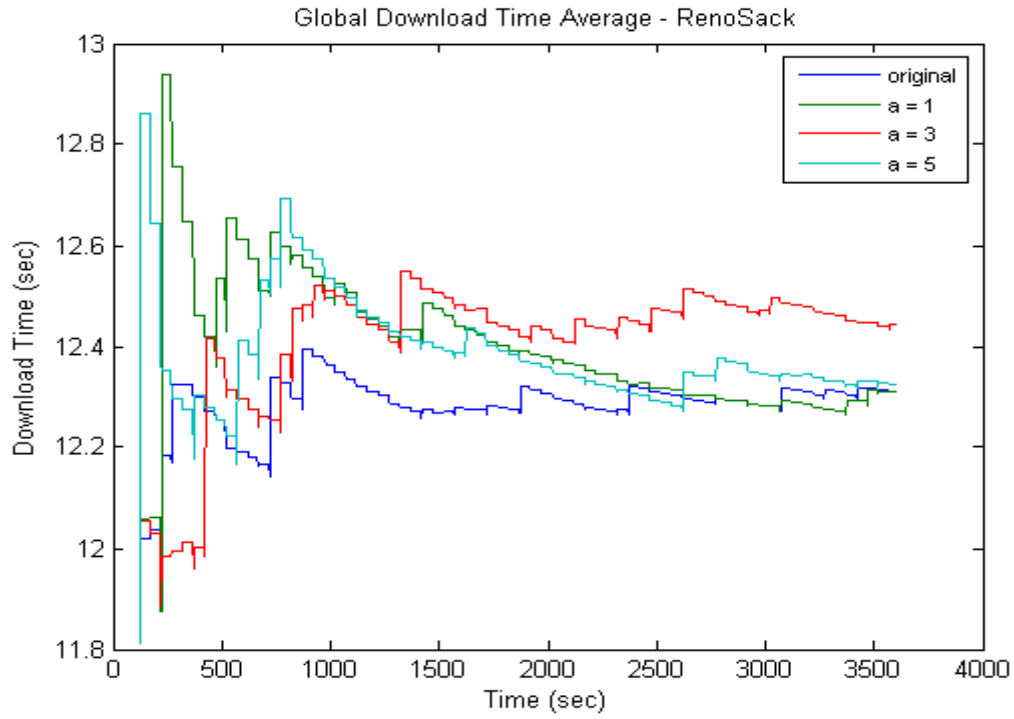

Figure 5.26: The global average of download time for 4SS scenario, utilizing Reno-SACK

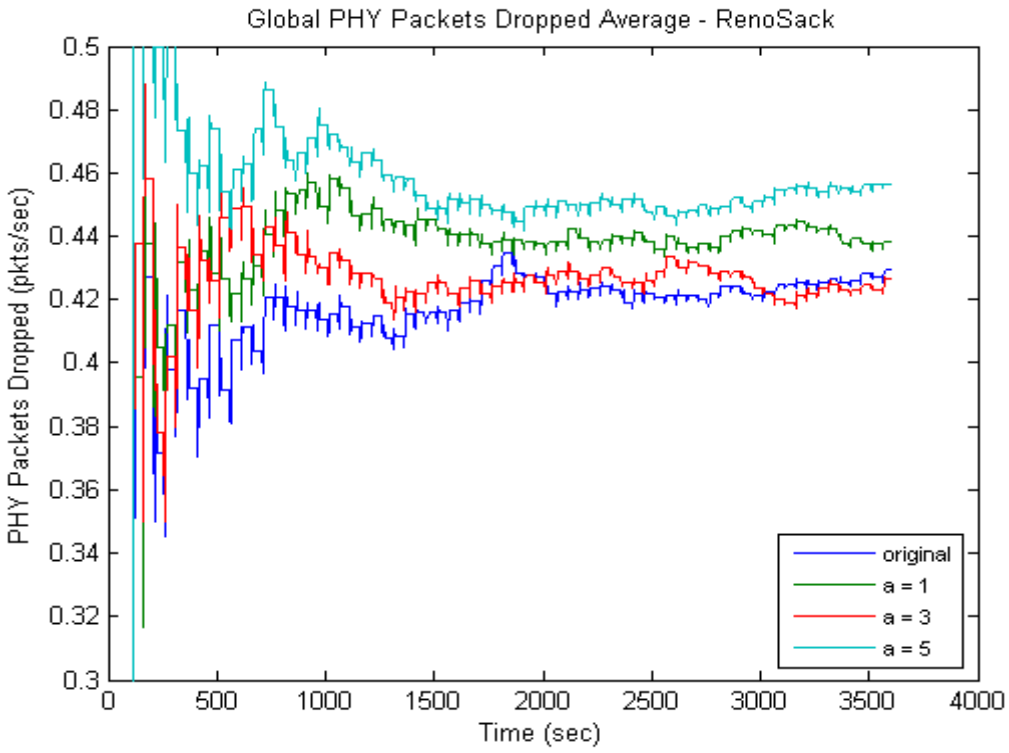

Figure 5.27: The global average of packets dropped for 4SS scenario, utilizing Reno-SACK

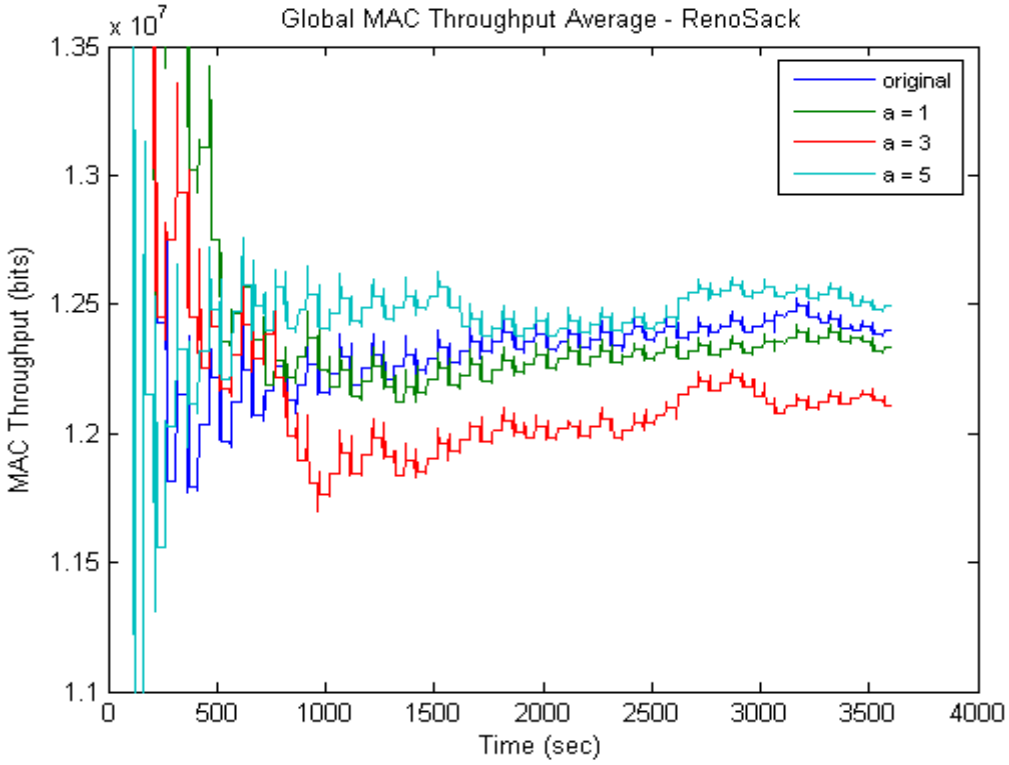

Figure 5.28: The global average of MAC throughput for 4SS scenario, utilizing Reno-SACK

Similar to the New Reno simulation, the proposed designs show a smaller delay at the MAC layer than the original, but the download times do no necessarily demonstrate the same improvement. Nevertheless, the graph of number of packets dropped shows less domination on the graph of file download time than in the Reno and New Reno scenarios. In addition, the performance of the file download time of the Reno-SACK combination is steadier (i.e. a smaller variation in the range of the y-axis) than in New Reno and Reno.

One of the conclusions that can be drawn from the three flavours of TCP when *N* is four is that the effect of the proposed algorithm becomes observable at the MAC layer delay. However, the file download time and throughput statistics are more complicate to anticipate. Nevertheless, if the improvement is significant and consistent at the low layer, it should be able to persist to higher layers.

As a packet is processed and delivered to the next layer in the hierarchy of the protocol stack, the performance metric measured in the next layer and layers afterwards becomes more difficult to enumerate. This is due to increasing complication on the measurement of a packet as the packet is being manipulated and influenced by mechanisms of each layer that the packet has visited. Therefore, the performance measured at the higher layer is more intricate in nature, in the sense that it exhibits effects of elements inherited from multiple layers. This consequence results in a problematic discernment on the effect of the algorithm, particularly for an algorithm implemented at the MAC layer.

On the other hand, though the MAC throughput is measured at the MAC layer, it is affected by many factors, such as error checking and packet drops.

Thus, throughput is also a complicated statistic to predict, especially in a wireless context.

5.3 Six Client Stations Scenario

In this section, simulations with six subscriber stations in the network are conducted. However, to avoid a tedious illustration of the simulated results for every TCP flavour, only key statistics will be presented to highlight the important discussions related to this thesis. Since the high-layer performance of TCP Reno is known to be significantly affected by the channel condition, it does not provide as clear indications on the effect of the proposed algorithm as New Reno and Reno-SACK combination. Furthermore, the Reno is generally not a recommended option to utilize in the wireless networks. Therefore, Reno is omitted in the detailed scenario-by-scenario illustration in this section, but overview graphs on the performances of Reno will still be included and discussed in the later section.

5.3.1 6SS – TCP New Reno

The plots for the MAC layer delay, file download time, number of packets dropped, and MAC throughput are presented in Figure 5.29 to Figure 5.32.

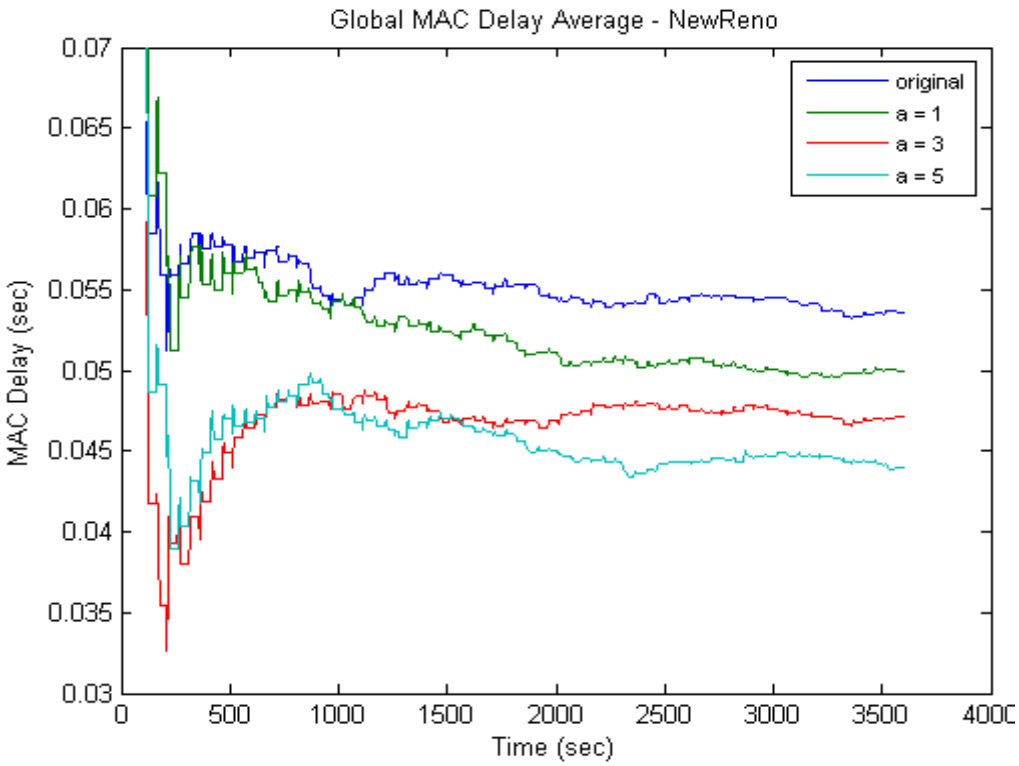

Figure 5.29: The global average of MAC delay for 6SS scenario, utilizing New Reno

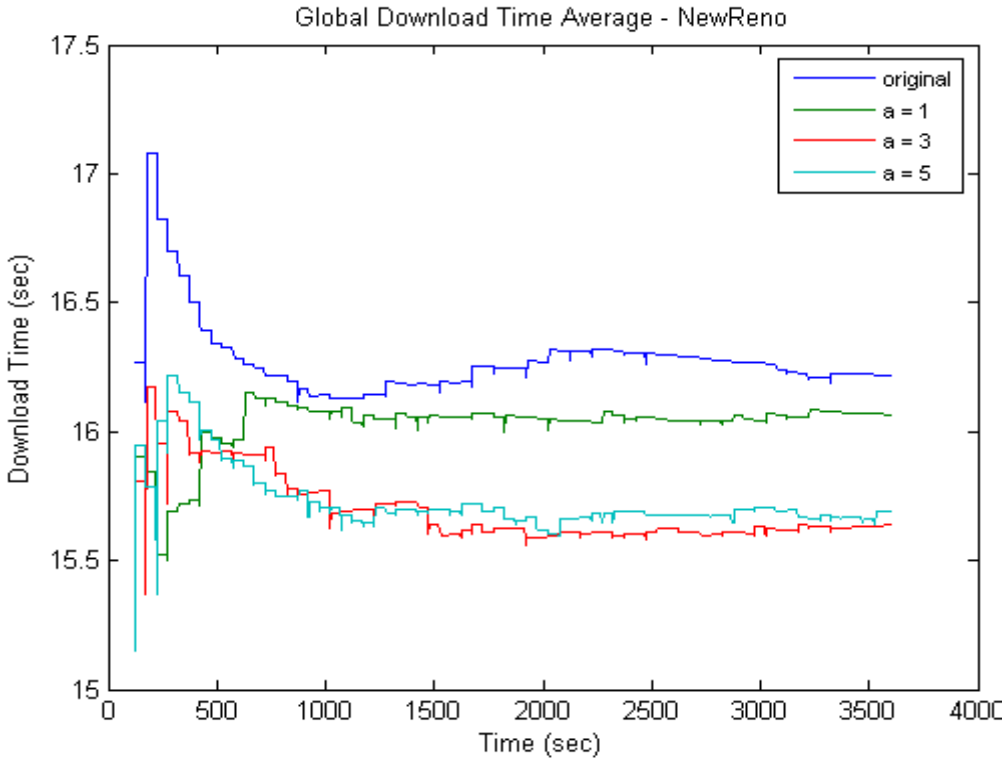

Figure 5.30: The global average of download time for 6SS scenario, utilizing New Reno

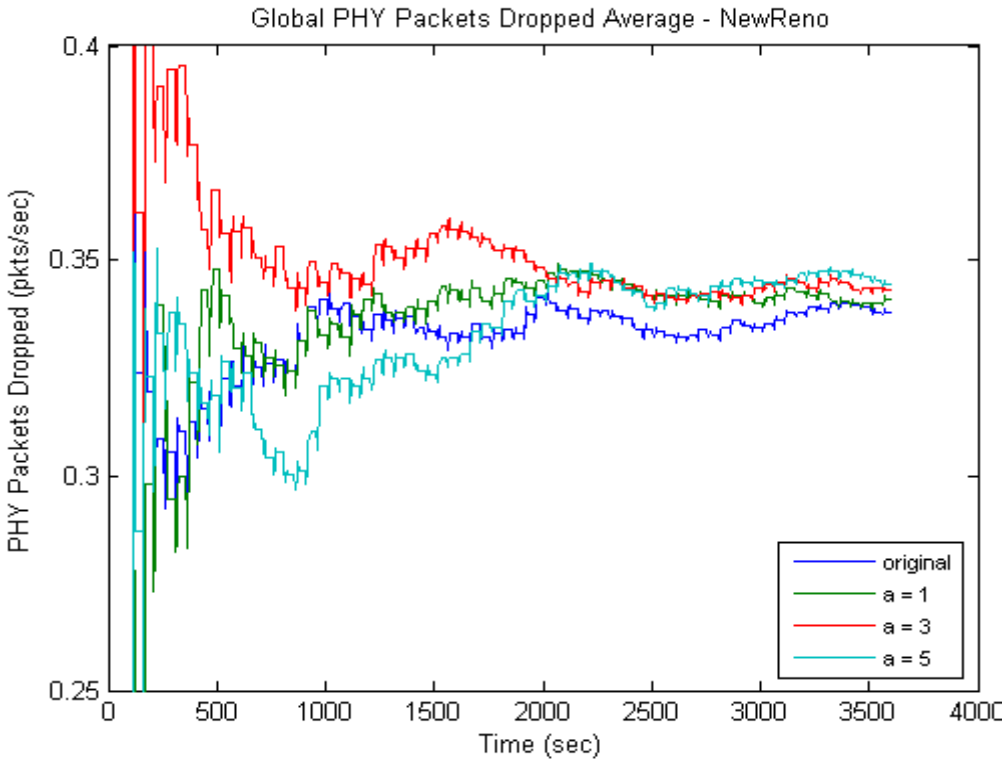

Figure 5.31: The global average of packets dropped for 6SS scenario, utilizing New Reno

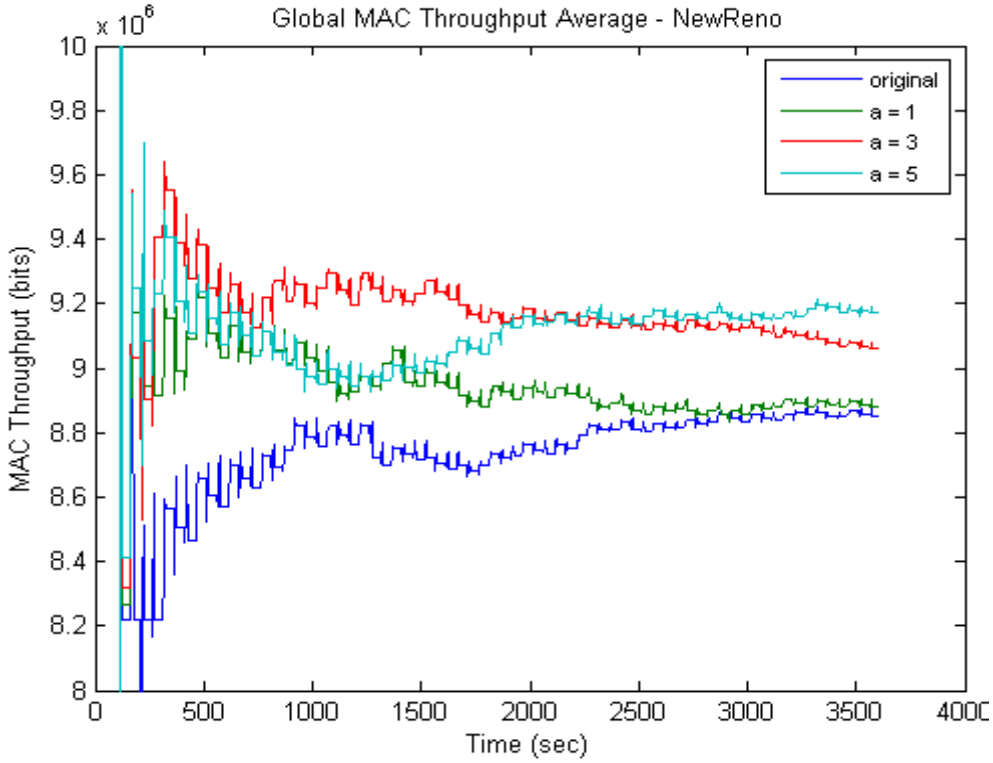

Figure 5.32: The global average of MAC throughput for 6SS scenario, utilizing New Reno

The graph of the MAC layer delay shows four well-spaced plots of the original and variations of the proposed design. The graph of the file download time begins to illustrate a steadier performance at the application layer, and it starts to reflect the gain at the MAC layer. More specifically, when *N*=6, the traffic intensity of the network begins to approach a moderate level, thus leading to a steady dequeuing process. The steady dequeuing process provokes the queue service rate to settle at the case (b) of the analysis in Section 3.4 more frequently. As a result, the weight of a queue becomes the significant factor in queue service rate, instead of the queue size. The condition of moderate traffic intensity allows the prediction of the analytical model built in Section 3.4 be more accurate. In fact, the observations from simulations comply with the analysis, in that the effect of the proposed algorithm is more evident when the number of stations in the network is higher.

5.3.2 6SS – TCP Reno & SACK

The plots of the MAC layer delay, file download time, PHY layer packet drop rate and MAC throughput of the 6SS scenario, utilizing combination of TCP Reno and SACK, are presented in Figure 5.33 to Figure 5.36.

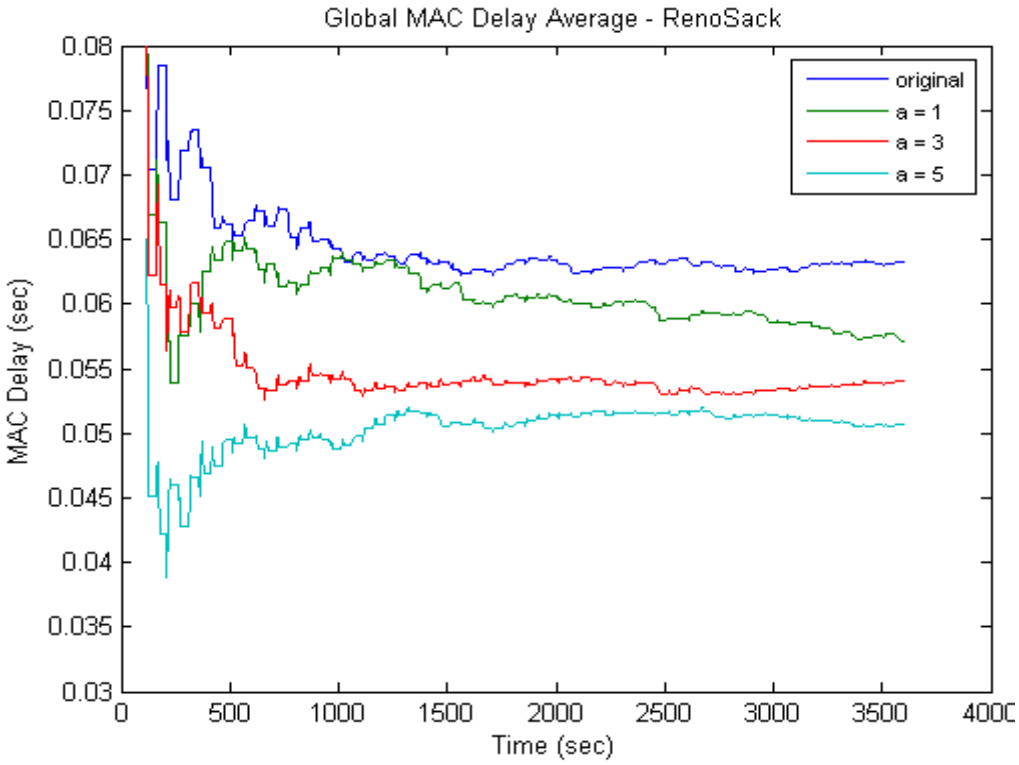

Figure 5.33: The global average of MAC delay for 6SS scenario, utilizing Reno-SACK

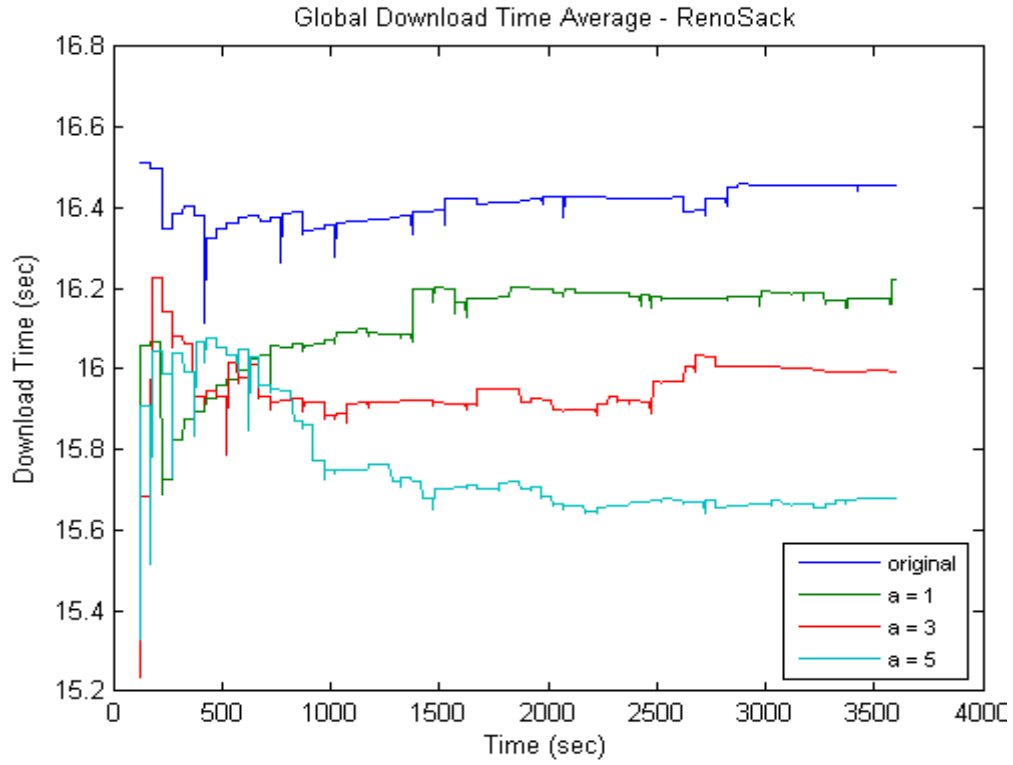

Figure 5.34: The global average of download time for 6SS scenario, utilizing Reno-SACK

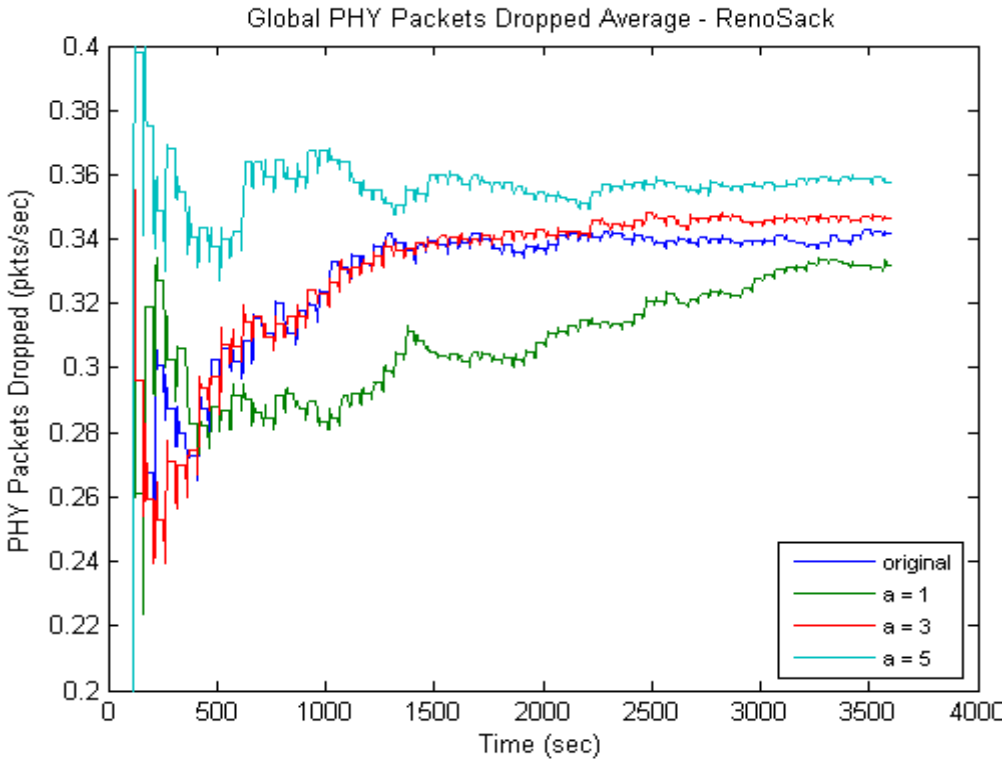

Figure 5.35: The global average of packets dropped for 6SS scenario, utilizing Reno-SACK

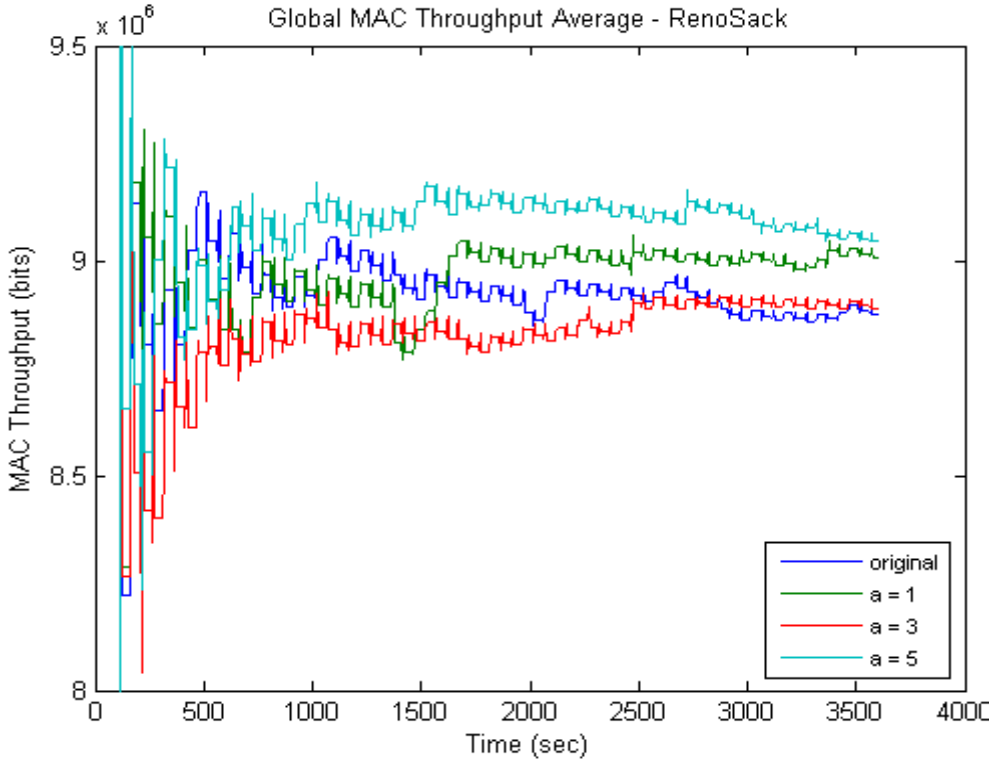

Figure 5.36: The global average of MAC throughput for 6SS scenario, utilizing Reno-SACK

The observations of the Reno-SACK scenario are similar to New Reno. The graph of the MAC layer delay consists of four distinct plots, and the plots are in the order of increasing weight-adjusting factor coefficients from top to bottom. Moreover, the plots of the file download time observed at the application layer are steadier, and better separated than in New Reno. At the same time, the fluctuating transient stage at the beginning of the simulation is shortened, and the range of the file download time is narrower. The resistance of Reno-SACK combination towards packet loss events is becoming observable. The plots of the number of packets dropped no longer dominate the trends or the order of the plots in the file download time graph.

The conclusion that can be drawn from the 6SS-scenarios is that the improvement at the MAC layer becomes more evident than in the 4SS and 2SS scenarios. The overall system reaches a steadier state as greater number of stations joined in the network, resulting in the traffic load is increased to a moderate level. The combination of a significant reduction of the MAC layer delay and steady traffic load leads to an improvement at the application layer. The number of packets dropped at the PHY layer shows even less influence on the average performance. Nevertheless, the flavour of TCP still plays an important role in delivering a decent performance at the application level.

5.4 Eight Client Stations Scenario

The same configurations are simulated with eight subscriber stations in the network. Like in the 6SS-senario, only simulations of New Reno and Reno-Sack combination are presented.

5.4.1 8SS – TCP New Reno

The graphs of the MAC layer delay and file download time are presented in Figure 5.37 and Figure 5.38. The number of packets dropped at the PHY layer and MAC throughput are illustrated in Figure 5.39 and Figure 5.40 respectively.

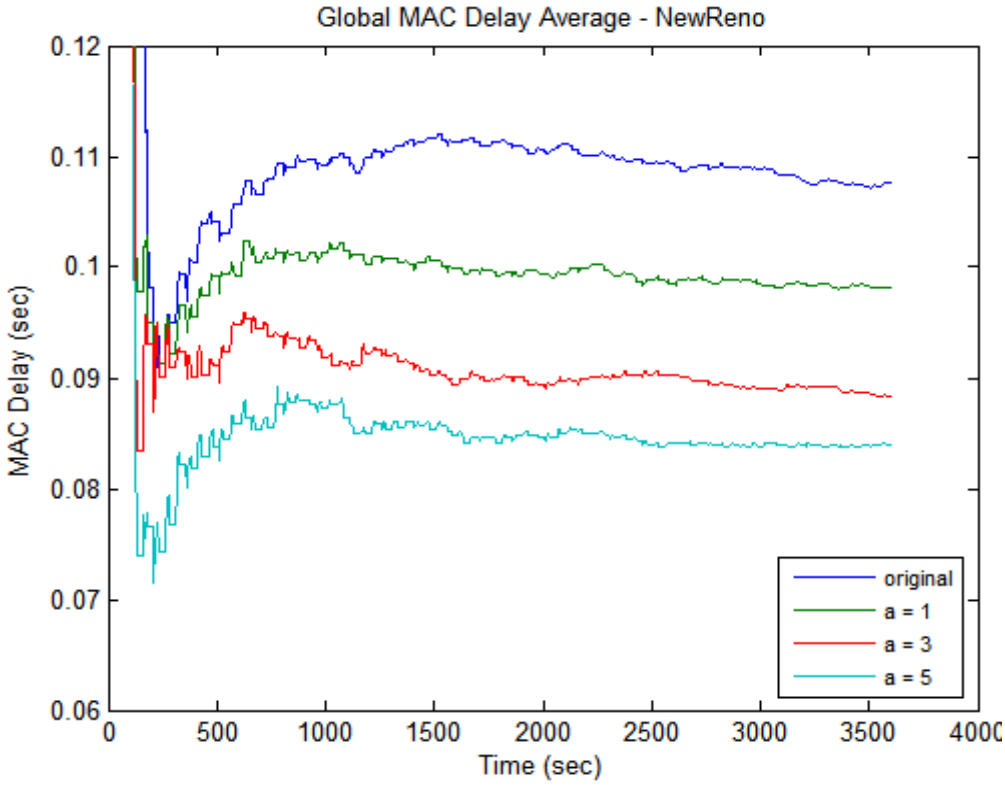

Figure 5.37: The global average of MAC delay for 8SS scenario, utilizing New Reno

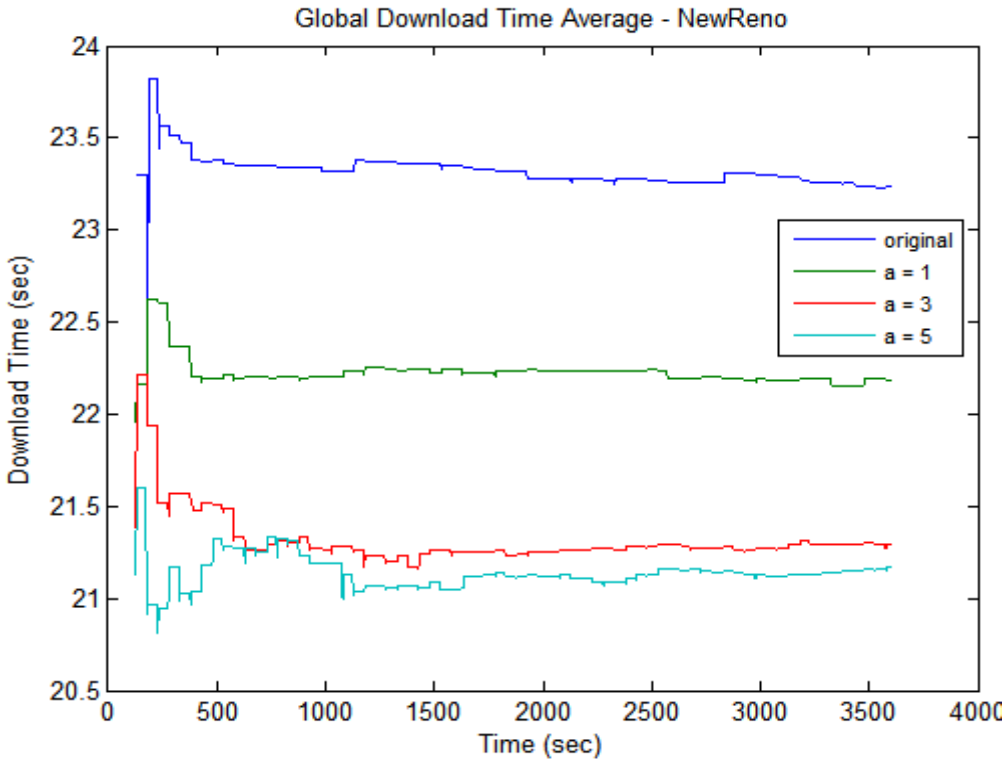

Figure 5.38: The global average of download time for 8SS scenario, utilizing New Reno

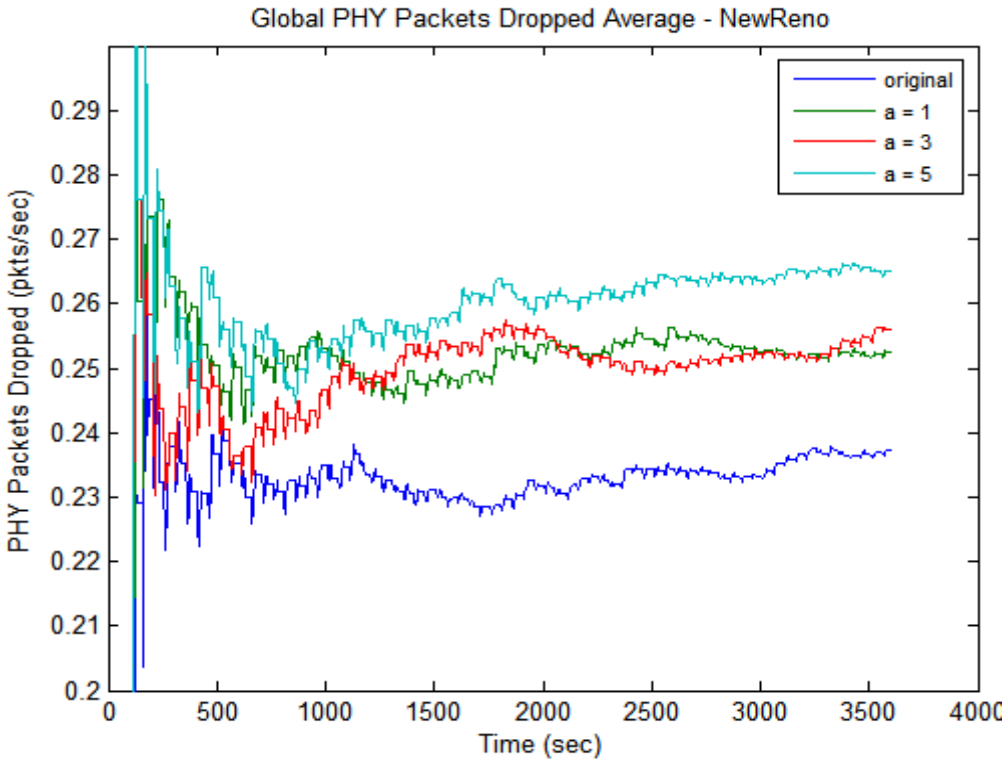

Figure 5.39: The global average of packets dropped for 8SS scenario, utilizing New Reno

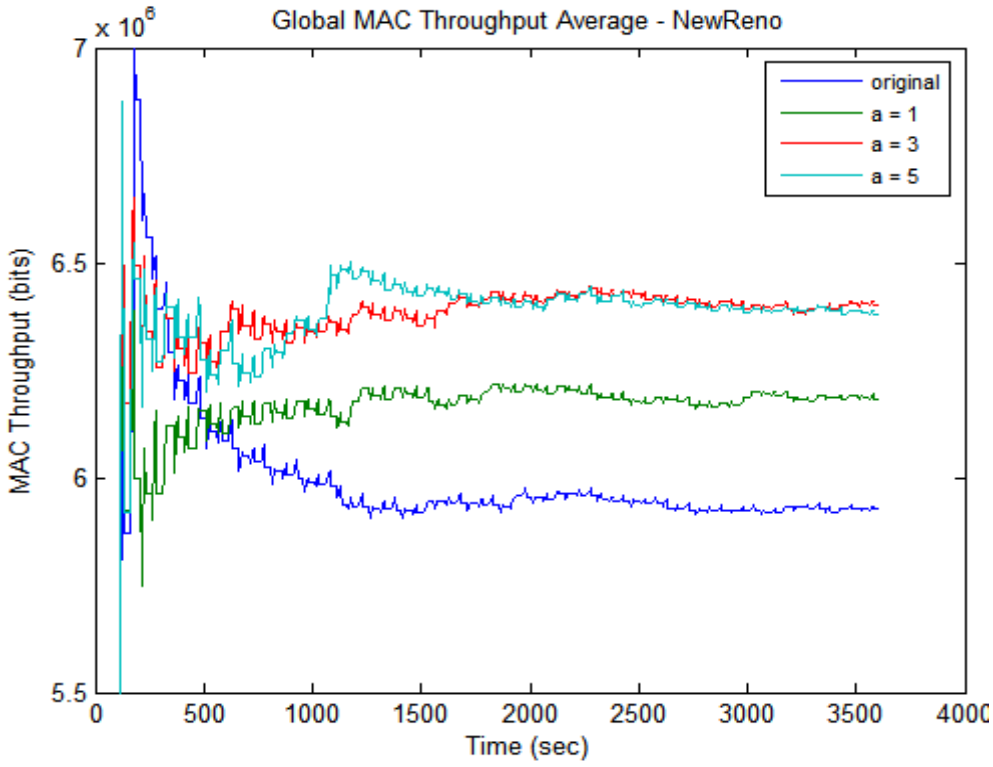

Figure 5.40: The global average of MAC throughput for 8SS scenario, utilizing New Reno

The improvement of the proposed design is distinctively illustrated in the graphs of the MAC layer delay and file download time. The initial transient stage of the 8SS-New Reno scenario is shorter than the 6SS-New Reno scenario. However, in both cases, the improvement is more evident with increasing values of weight-adjusting factor coefficients. Moreover, each plot of the file download time is spaced further apart in the 8SS-New Reno scenario than in the 6SS-New Reno scenario, such that each design fluctuates mostly within its own range of y-values. In addition, the improvement is persistent, regardless of the drop rate at the physical link. Finally, the throughput plots show improvements across the designs when *N* is eight.

5.4.2 8SS – TCP Reno and SACK

The simulation results of eight subscriber stations, which utilize Reno and SACK combination, are presented in this sub-section. The plots of the MAC layer delay, file download time, number of packets dropped, and throughput are illustrated in Figure 5.41 to Figure 5.44.

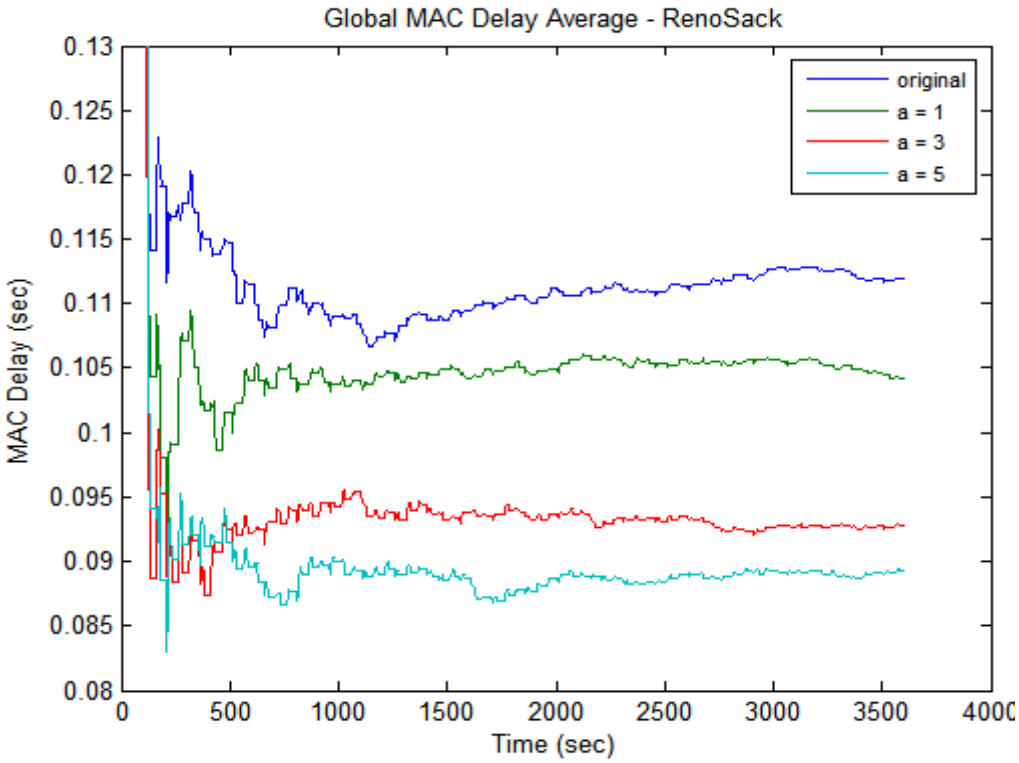

Figure 5.41: The global average of MAC delay for 8SS scenario, utilizing Reno-SACK

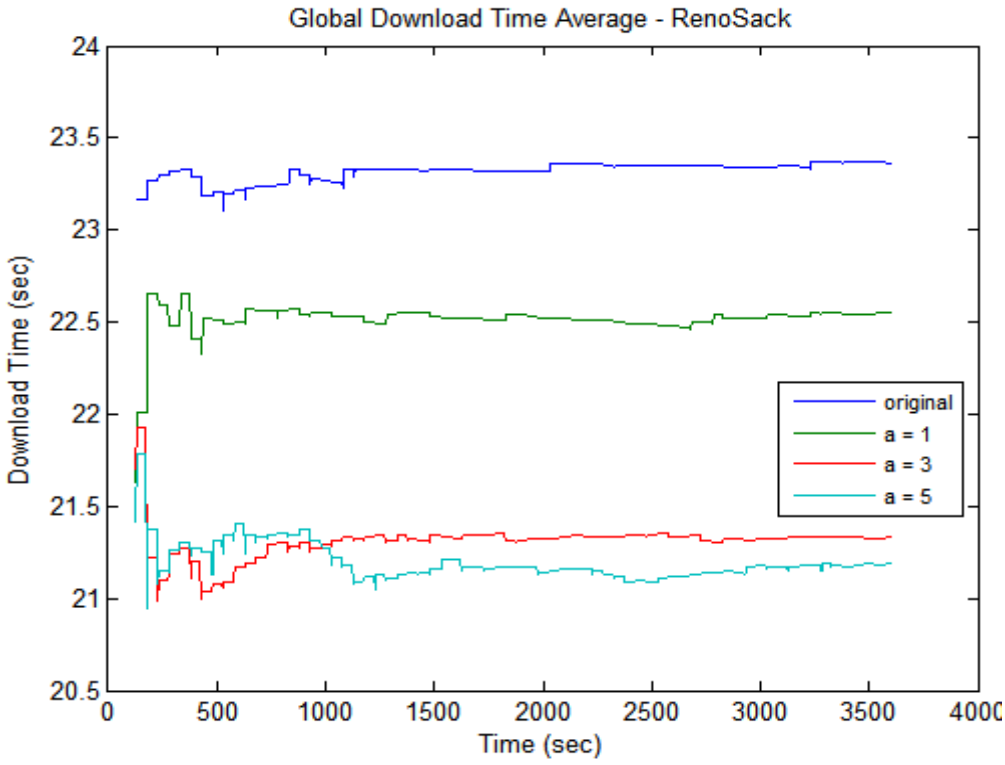

Figure 5.42: The global average of download time for 8SS scenario, utilizing Reno-SACK

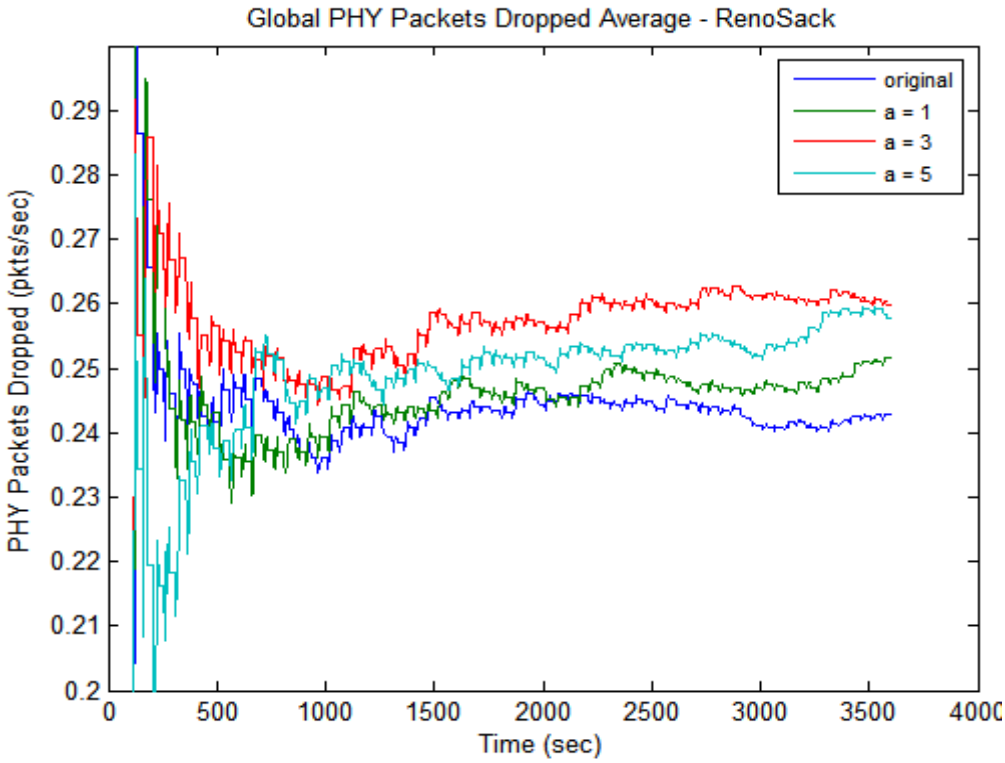

Figure 5.43: The global average of packets dropped for 8SS scenario, utilizing Reno-SACK

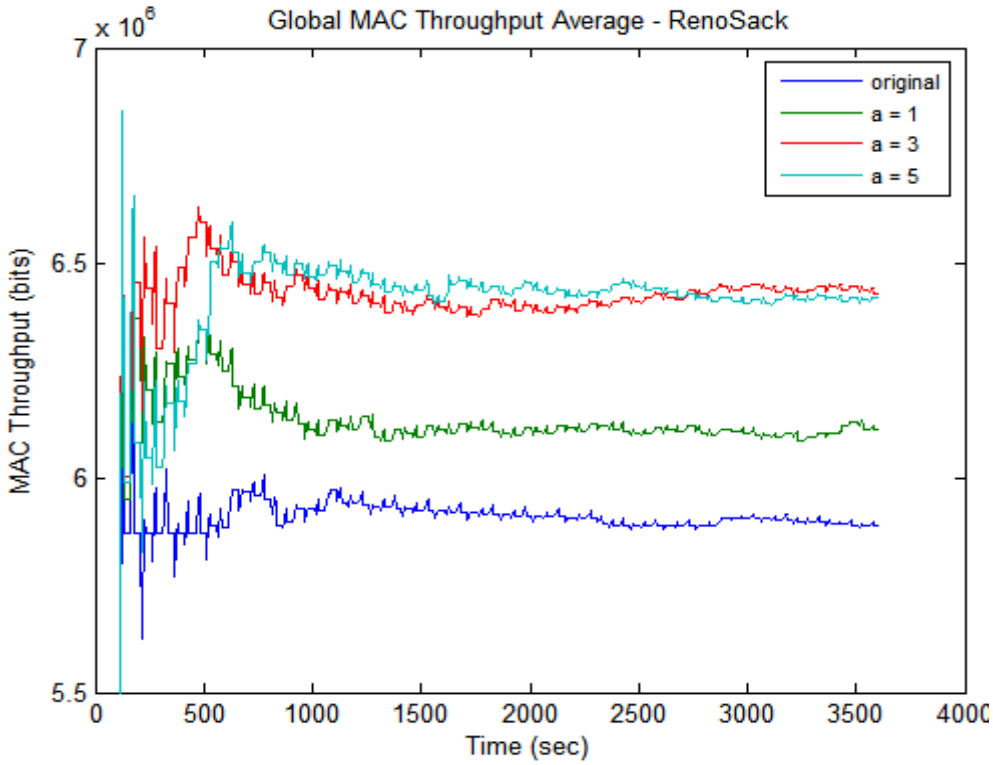

Figure 5.44: The global average of MAC throughput for 8SS scenario, utilizing Reno-SACK

The Reno-SACK combination demonstrates similar results as in the 8SS-New Reno case. More specifically, the improvements of the proposed algorithm are evident in the MAC layer delay, file download time and throughput. The improvement compared to the original design increases with respect to the weight-adjusting factor coefficient, but not the growth of the improvement in between designs. In fact, the improvement from *a*=3 to *a*=5 is less than that from *a*=1 to *a*=3. In short, the benefit of the increasing weight-adjusting factor coefficient diminishes beyond a certain value. This observation complies with the analysis in Section 3.4, and suggests that the proposed scheme has a maximum performance benefit at a certain coefficient value. An overly aggressive (i.e. large)

weight-adjusting factor coefficient may result in a decaying and unfair performance.

Based on the simulation results illustrated in the two, four, six and eight subscriber stations scenarios, the proposed algorithm shows a better performance as the number of stations in the network is sufficiently large. The traffic intensity of the system reaches a moderate level, resulting in a steady operation in queue service and the resulting performance. A large weight-adjusting factor coefficient helps to attain a better outcome, but the improvement is limited to a certain extent.

Another conclusion that can be drawn from the simulations is that the MAC layer delay is the most sensitive performance measurement that reflects the effect of the proposed scheme. In particular, the improvement of the MAC layer delay is observed starting from the 4SS-scenarios, whereas the improvement of the download time is observed in the 6SS-scenarios and beyond. The improvement of throughput is also observed in 6SS-scenarios and beyond, but it only becomes more evident in 8SS-scenarios. This observation is reasonable since the MAC layer delay is a statistic, which simply measures the end-to-end delay of every packet received. In contrast, both file download time and throughput are affected by complicated mechanisms, such as error checking, packet drops and timeout events, which make them more difficult to reflect the effect of the proposed design.

The 10-SS, 12-SS and 15-SS scenario simulations were also conducted, but they are not illustrated in detail as in the 2SS, 4SS, 6SS and 8SS-scenarios.

Instead, the statistics are plotted against various values of *N*, as it is shown in the next section.

5.5 Performance with respect to *N*

This sub-section provides an overview of the results illustrated in Section 5.1 to Section 5.4, and in addition, simulation results of 10-SS, 12-SS and 15-SS scenarios are included. The performance metrics such as the MAC layer delay, file download time, and MAC throughput are plotted against various values of *N*. These figures portray visualization of the effect of the proposed scheme with respect to different levels of traffic intensity in the network. The order of the presentation is arranged in accordance with the TCP flavours as before.

5.5.1 The MAC Layer Delay vs. Number of Stations

The global average of MAC layer delay with various numbers of subscriber stations, which utilizes Reno as the TCP flavour, is plotted in Figure 5.45.

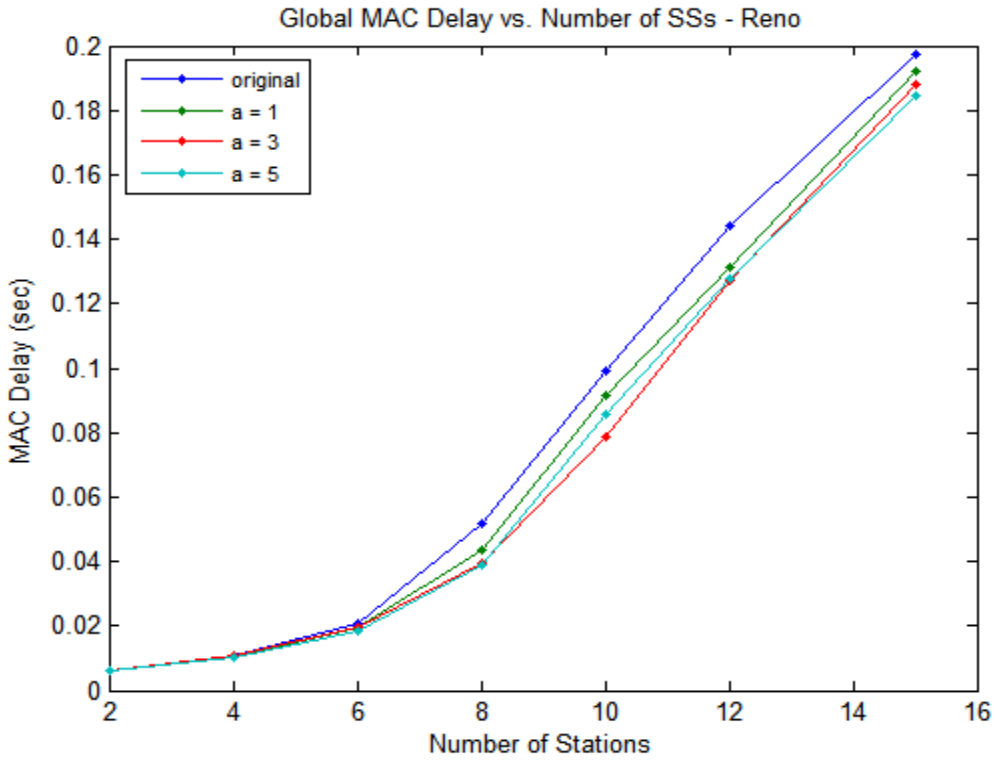

Figure 5.45: MAC delay vs. number of stations, utilizing Reno

The number of stations that were simulated includes two, four, six, eight, ten, twelve and fifteen, and the y-axis is the final value of the global average of the MAC layer delay. The figure shows an increasing trend of the MAC layer delay with respect to the number of stations. This is anticipated as the resources of the BS are shared by more stations, thus resulting in an increasing queue delay. When *N* equals to two, the MAC layer delay is small but indistinguishable. In fact, the MAC layer delay of the proposed design is worse than the original when *N* equals to two, as illustrated in Section 5.1.1. Though the improvement of the MAC layer delay is observed when *N* equals to four in Reno, as demonstrated in Section 5.2.1, the improvement is small and not significant. When *N* is greater or equals to eight, the plots of the proposed designs start to

pull away from the original design. Table 5-1 provides the detailed information on the percentage difference of each proposed design when comparing to the original design.

Table 5-1: The percentage differences of the global average delay at the MAC layer of each proposed design compared to the original design, utilizing Reno

N	**a = 1**	**a = 3**	**a = 5**
2	0.21%	0.12%	0.33%
4	-5.41%	-2.26%	-9.20%
6	-5.56%	-3.07%	-8.91%
8	-16.02%	-24.46%	-25.41%
10	-8.13%	-20.97%	-13.70%
12	-8.84%	-11.79%	-11.19%
15	-2.65%	-4.66%	-6.46%

When *N* is two, the percentages are positive. This indicates that the MAC layer delays of those scenarios are higher than the original design. When *N* is greater than two, the percentage differences become negative, indicating a smaller MAC layer delay of the proposed designs in those scenarios. The maximum reduction is 25.41%, which occurs when *N* is eight, and *a* is five. Furthermore, the table again demonstrates that the gain of the proposed design increases with respect to *N* and *a*, but the gains are limited at certain bound.

The MAC layer delay versus the number of stations in the network, which employs New Reno as the TCP flavour, is illustrated in Figure 5.46.

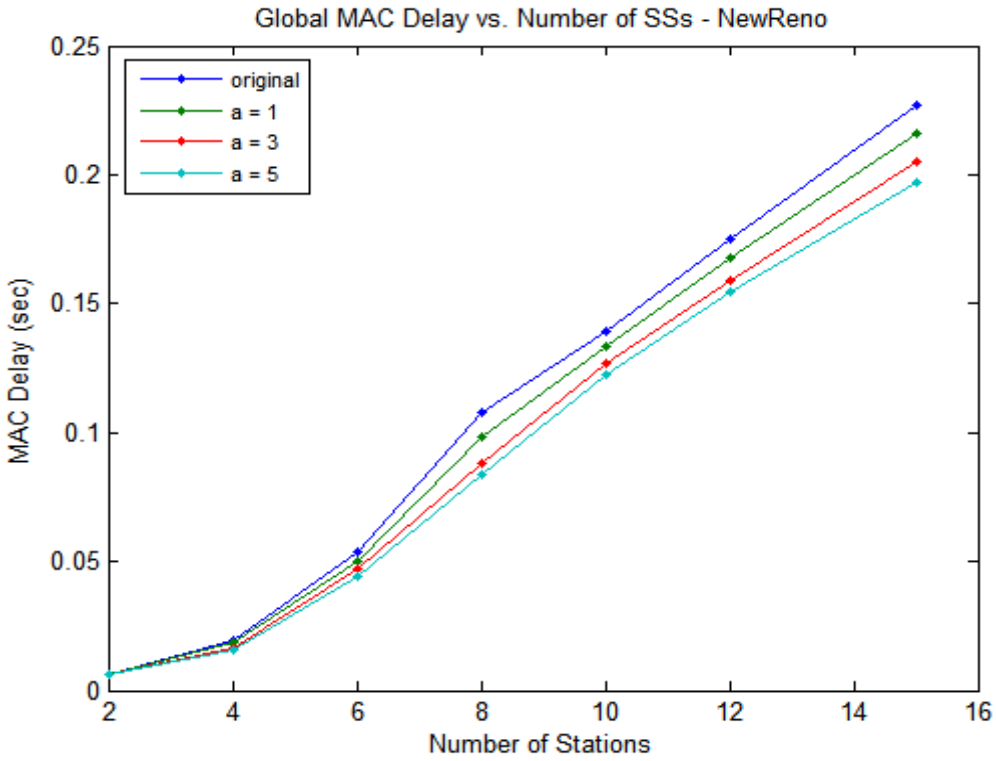

Figure 5.46: MAC delay vs. number of station, utilizing New Reno

The New Reno scenario also exhibits an increasing trend in the plots of MAC layer delay with respect to the number of stations. Nevertheless, the plots are more distinguishable starting when *N* is six, or even when *N* is four. For *N* greater than four, the MAC layer delays of the proposed designs are consistently lower than the original, and the differences become more evident when the number of stations exceeds eight. In addition, the gain of the proposed algorithm grows with respect to the increasing *a*. A detailed comparison on the improvement of each data point is listed in Table 5-2.

Table 5-2: The percentage differences of the global average delay at the MAC layer of each proposed design compared to the original design, utilizing New Reno

N	**a = 1**	**a = 3**	**a = 5**
2	0.46%	0.49%	0.78%
4	-4.59%	-16.30%	-21.13
6	-6.79%	-12.14%	-17.96%
8	-8.70%	-17.87%	-21.93%
10	-4.66%	-8.93	-12.26%
12	-3.87%	-9.20%	-11.51%
15	-5.02%	-9.90%	-13.46%

When *N* equals to fifteen, the *a*=1 design reduces the delay of the original design by approximately 5%, and the *a*=5 design delivers a reduction of 13.46%. Nevertheless, the maximum reduction is 21.93%, which occurs when *N* is eight and *a* is five. The MAC layer delay of the scenarios utilizing the Reno-SACK combination is presented in Figure 5.47.

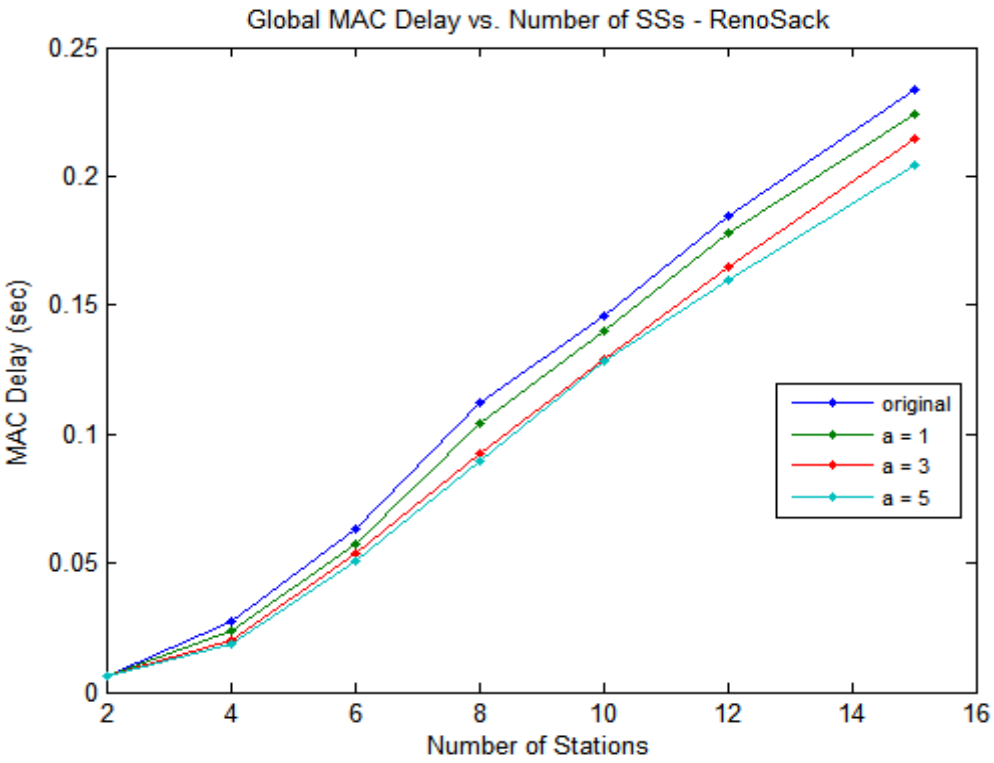

Figure 5.47: MAC delay vs. number of station, utilizing Reno-SACK

The proposed designs also exhibit consistent reductions in the MAC layer delay in the Reno-SACK combination. When *N* equals to ten, the gains in the *a*=3 and *a*=5 cases are not differentiable. When the number of stations increases to beyond ten, the plot of *a*=5 design begins to move apart from the plot of *a*=3. This behaviour can be explained by the discussion in Section 3.4, in which a greater *a* requires a greater *N* in order to offset the effect of the coefficient at the denominator. The *a*=1 delivers a reduction of 4% in the MAC layer delay when *N* is fifteen, while in the same scenario, the *a*=5 design reduces the original design by 12.6%. Table 5-3 lists the detailed reductions in the MAC layer delay of each scenario when employing Reno-SACK combination as the TCP flavour.

Table 5-3: The percentage differences of the global average delay at the MAC layer of each proposed design compared to the original design, utilizing Reno-SACK

N	a = 1	a = 3	a = 5
2	0.47%	0.38%	0.23%
4	-12.90%	-26.46%	-32.24%
6	-9.74%	-14.62%	-20.04%
8	-6.90%	-17.12%	-20.22%
10	-4.02%	-11.57%	-11.89%
12	-3.53%	-10.64%	-13.27%
15	-4.01%	-8.06%	-12.63%

5.5.2 FTP File Download Time vs. Number of Stations

This sub-section provides an overview of the performance of file download time at the application layer with respect to various numbers of stations in the network. The figures are illustrated in the same fashion as in the MAC layer delay section. The graphs of the Reno and New Reno scenarios are presented in Figure 5.48 and Figure 5.49 respectively, and graph of the Reno-SACK combination is shown in Figure 5.50. Each figure is followed by a table to provide the detailed information on the percentage difference of each proposed design compared to the original one.

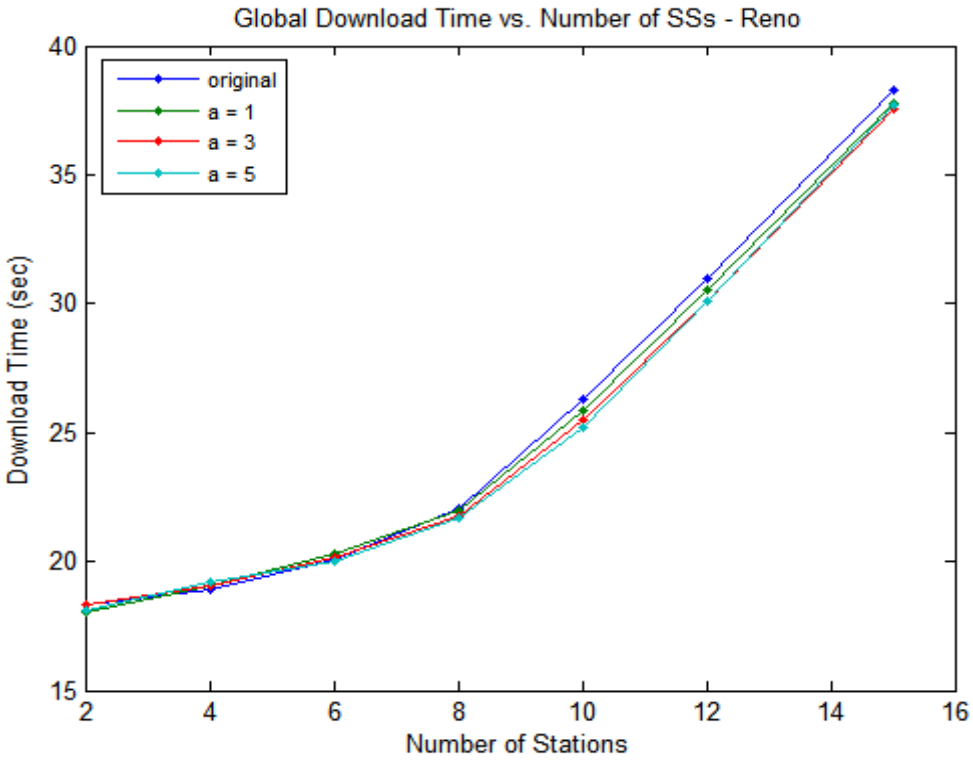

Figure 5.48: The file download time vs. number of stations, utilizing Reno

Table 5-4: The percentage differences of the global average of the file download time of each proposed design compared to the original design, utilizing Reno

N	a = 1	a = 3	a = 5
2	-1.58%	0.21%	-1.08%
4	0.81%	0.73%	1.43%
6	1.01%	0.49%	-0.30%
8	-0.25%	-1.42%	-1.79%
10	-1.66%	-3.10%	-4.03%
12	-1.40%	-2.75%	-2.70%
15	-1.39%	-1.91%	-1.63%

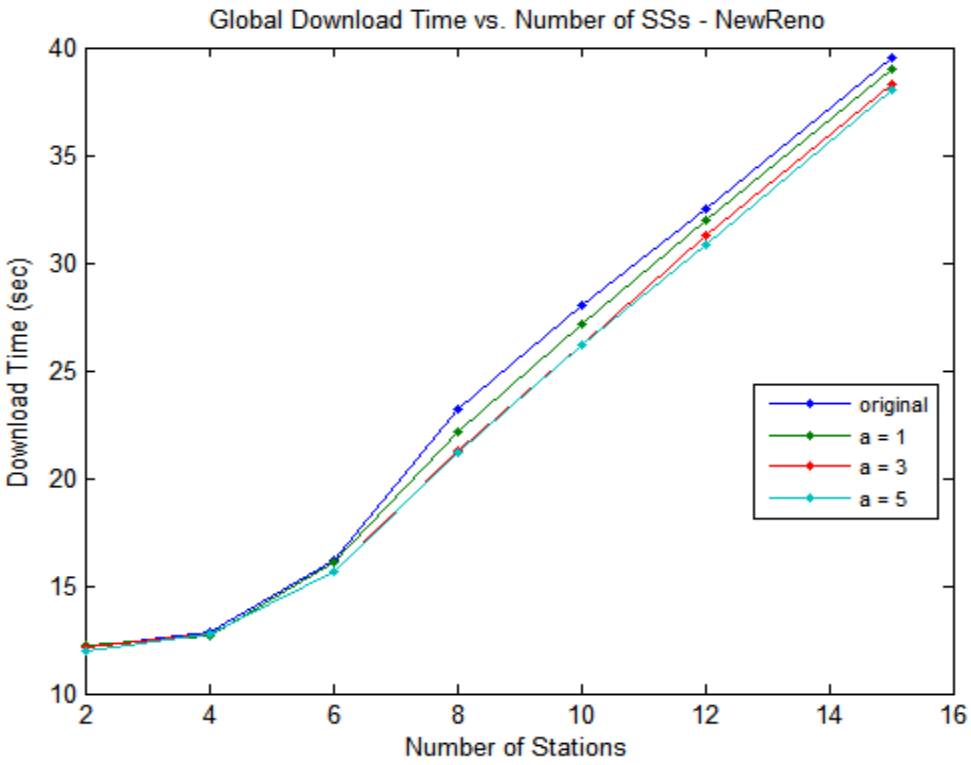

Figure 5.49: The file download time vs. number of station, utilizing New Reno

Table 5-5: The percentage differences of the global average of the file download time of each proposed design compared to the original design, utilizing New Reno

N	**a = 1**	**a = 3**	**a = 5**
2	0.23%	-0.54%	-2.03%
4	-1.28%	-0.53%	-1.04%
6	-0.92%	-3.53%	-3.21%
8	-4.52%	-8.37%	-8.88%
10	-3.09%	-6.51%	-6.77%
12	-1.89%	-3.96%	-5.20%
15	-1.42%	-3.09%	-3.77%

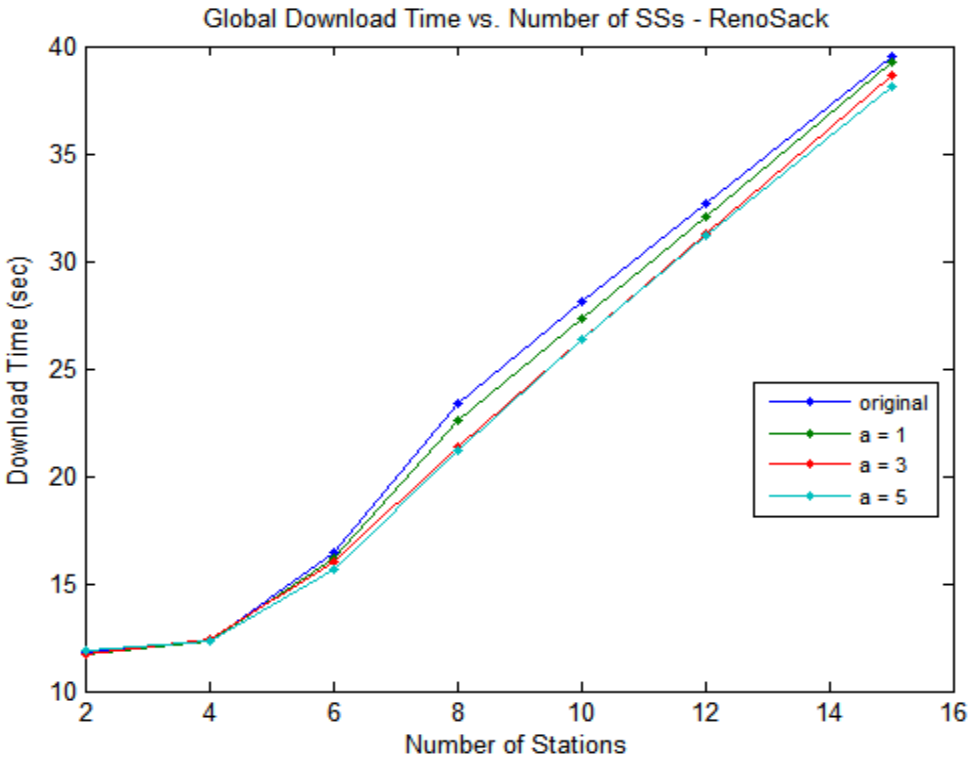

Figure 5.50: FTP file download time vs. number of station, utilizing Reno and SACK

Table 5-6: The percentage differences of global average of the file download time of each proposed design compared to the original design, utilizing Reno-SACK

N	**a = 1**	**a = 3**	**a = 5**
2	-0.13%	-0.41%	0.84%
4	-0.01%	1.08%	0.11%
6	-1.43%	-2.82%	-4.72%
8	-3.49%	-8.69%	-9.30%
10	-2.84%	-5.97%	-6.01%
12	-1.95%	-4.15%	-4.54%
15	-0.70%	-2.28%	-3.61%

The file download time is similar to the MAC layer delay, which exhibits a rising trend when the number of stations in the network increases. Nevertheless,

the reductions in the file download time of the proposed designs are not as extraordinary as in the MAC layer delay. In particular, the *a*=3 and *a*=5 are not differentiable until *N* is twelve in New Reno and fifteen in Reno-SACK combination. Furthermore, the download times of the Reno scenarios are even more difficult to differentiate than New Reno and Reno-SACK combination. As previously mentioned, download time is a more complicated statistic to anticipate, which incorporates many qualities of the mechanisms at the lower layers. In particular, Reno suffers the worst among the three flavours because Reno is the most vulnerable scheme of the three when encountering packet drops in the physical channel.

The percentage gains of the file download time are smaller than 10% in all scenarios, which is less than the MAC layer delay. When *N* is equal to fifteen, the *a*=1 performs 1.42% better than the original in New Reno, and 0.7% better than the original in Reno-SACK combination. In comparison, the *a*=5 design delivers a reduction of 3.77% in New Reno, and 3.6% in the Reno-SACK combination when *N* is fifteen. Therefore, the coefficient of the weight-adjusting factor still makes a difference. Nevertheless, the largest reduction in the file download time happens when *N* equals to eight, where the reduction of the *a*=5 design in the Reno-SACK scenario is 9.3%.

5.5.3 MAC Throughput vs. Number of Stations

The throughput plots with different numbers of stations in the network are presented in this sub-section. In contrast to the delay graphs, the MAC throughput decreases as the number of stations grows. Since the throughput is a

reducing statistic with increasing *N*, the figure is cropped at *N* equal to eight to better illustrate the comparison of the designs at high values of *N* (i.e. when the system performance are more stable with steady traffic load). The throughput graphs that utilize TCP Reno and New Reno are illustrated in Figure 5.51 and Figure 5.52 respectively. The throughput graph of the Reno-SACK combination is presented in Figure 5.53. The detailed comparison of each proposed design to the original is provided in a table after each figure.

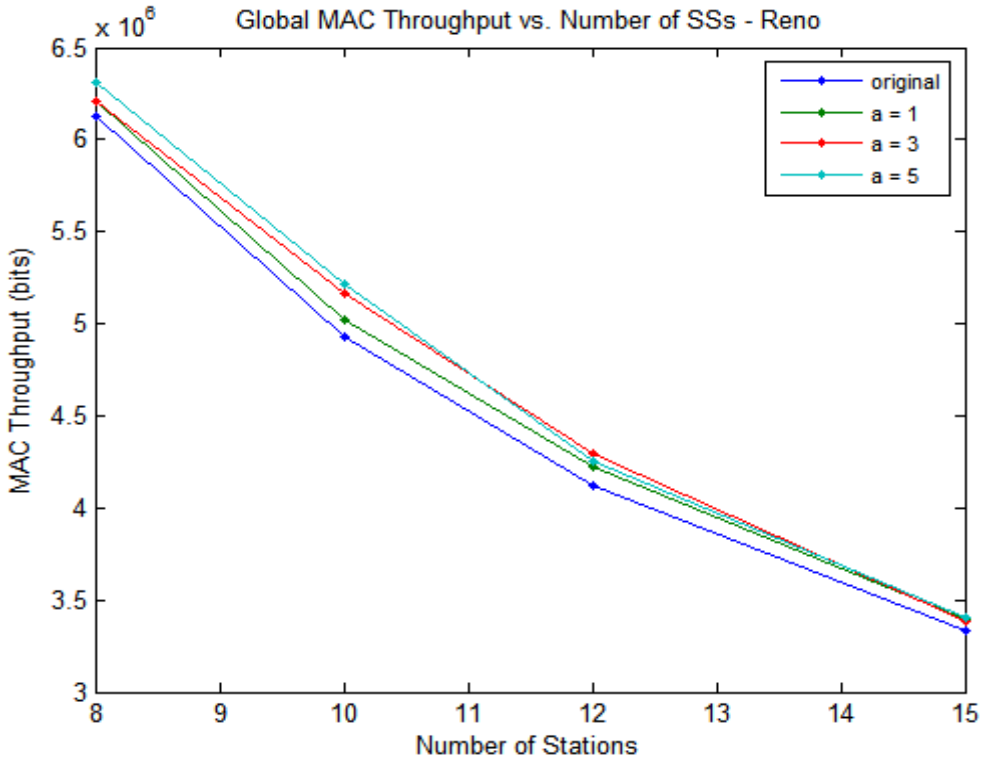

Figure 5.51: MAC throughput vs. number of station, utilizing Reno

Table 5-7: The percentage differences of global average of the MAC throughput of each proposed design compared to the original design, utilizing Reno

N	a = 1	a = 3	a = 5
2	3.32%	-0.76%	1.00%
4	-1.18%	0.55%	-0.97%
6	0.37%	1.70%	0.78%
8	1.27%	1.36%	2.99%
10	2.06%	4.92%	5.93%
12	2.48%	4.23%	3.22%
15	1.85%	1.62%	2.31%

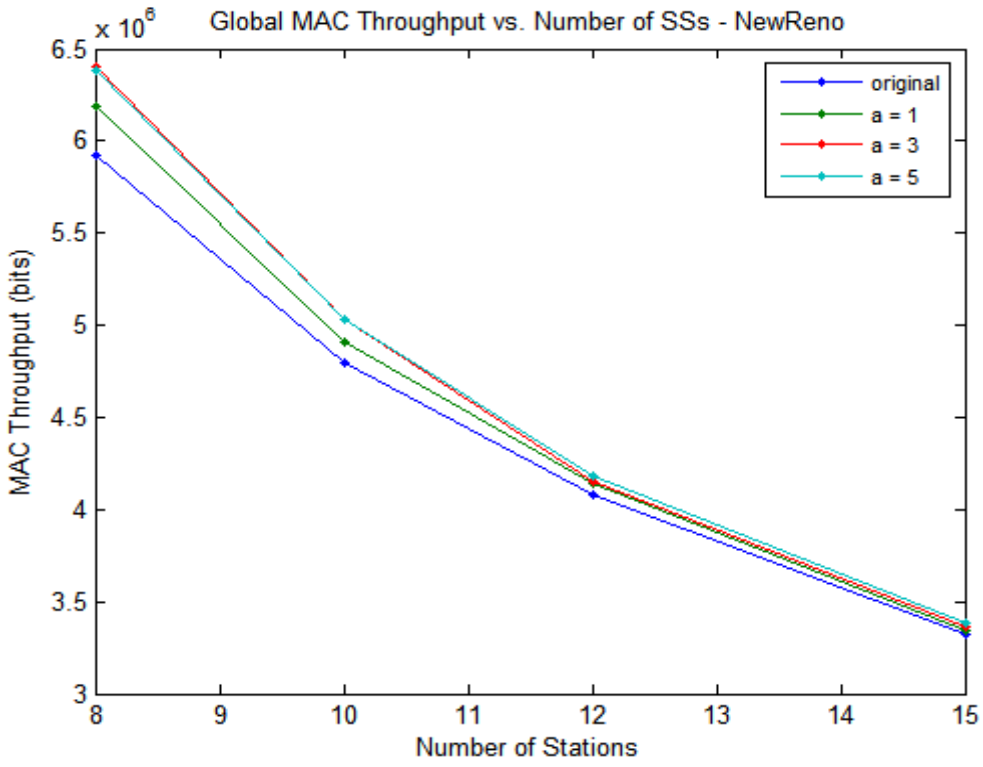

Figure 5.52: MAC throughput vs. number of station, utilizing New Reno

Table 5-8: The percentage differences of global average of the MAC throughput of each proposed design compared to the original design, utilizing New Reno

N	a = 1	a = 3	a = 5
2	3.74%	4.53%	4.49%
4	-1.30%	-0.55%	-0.36%
6	0.32%	2.37%	3.61%
8	4.34%	8.04%	7.63%
10	2.33%	4.88%	4.94%
12	1.41%	1.77%	2.33%
15	0.49%	1.10%	1.79%

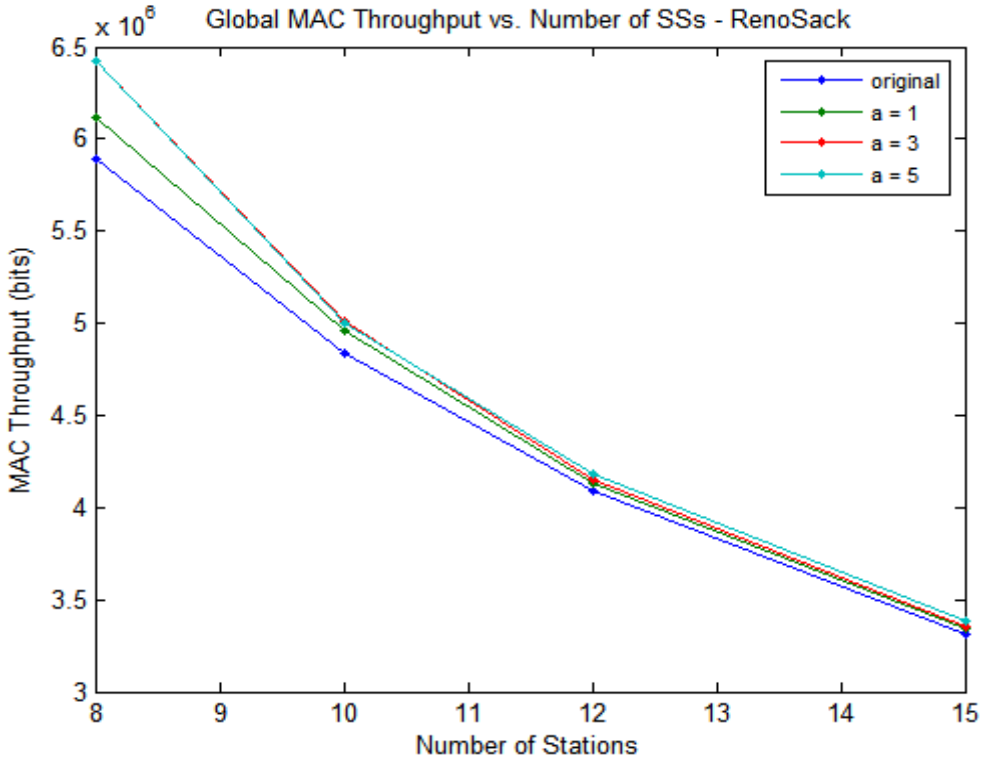

Figure 5.53: MAC throughput vs. number of station, utilizing Reno-SACK

Table 5-9: The percentage differences of global average of MAC throughput of each proposed design compared to the original design, utilizing Reno-SACK

N	a = 1	a = 3	a = 5
2	0.66%	-0.52%	-1.59%
4	-0.53%	-2.38%	0.76%
6	1.48%	0.14%	1.93%
8	3.77%	9.14%	8.99%
10	2.53%	3.66%	3.48%
12	0.92%	1.49%	2.18%
15	0.85%	1.39%	2.08%

The graphs indicate that the proposed design produces a higher throughput than the original design. However, from the tables, some scenarios of the proposed design deliver worse throughput than the original when *N* is small. When *N* is larger, the improvement becomes more distinct. In particular, the improvement when *N* equals to eight is 8.04% in New Reno for the *a*=3 design, and 9.14% in Reno-SACK for the *a*=3 design. Nevertheless, the plots of *a*=3 and *a*=5 appear to be indistinguishable, and become more difficult to observe when *N* is larger. To better illustrate the throughput when *N* is large, the slightly enlarged versions of the above graphs are shown in Figure 5.54 to Figure 5.56.

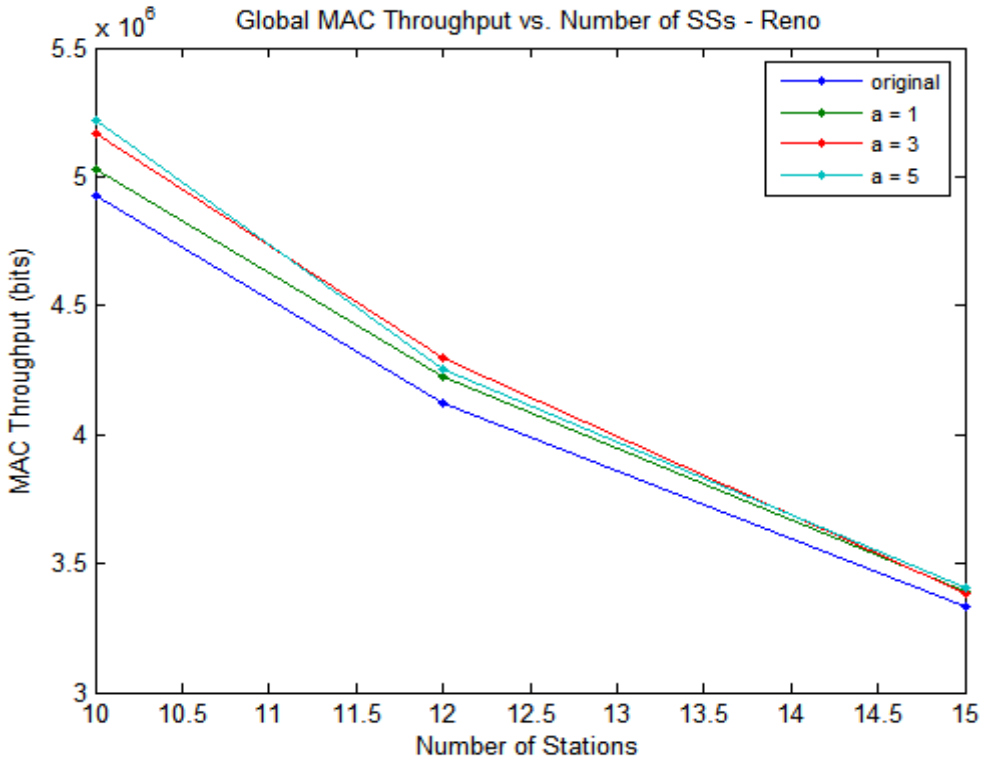

Figure 5.54: MAC throughput vs. number of station, utilizing Reno (Zoom In)

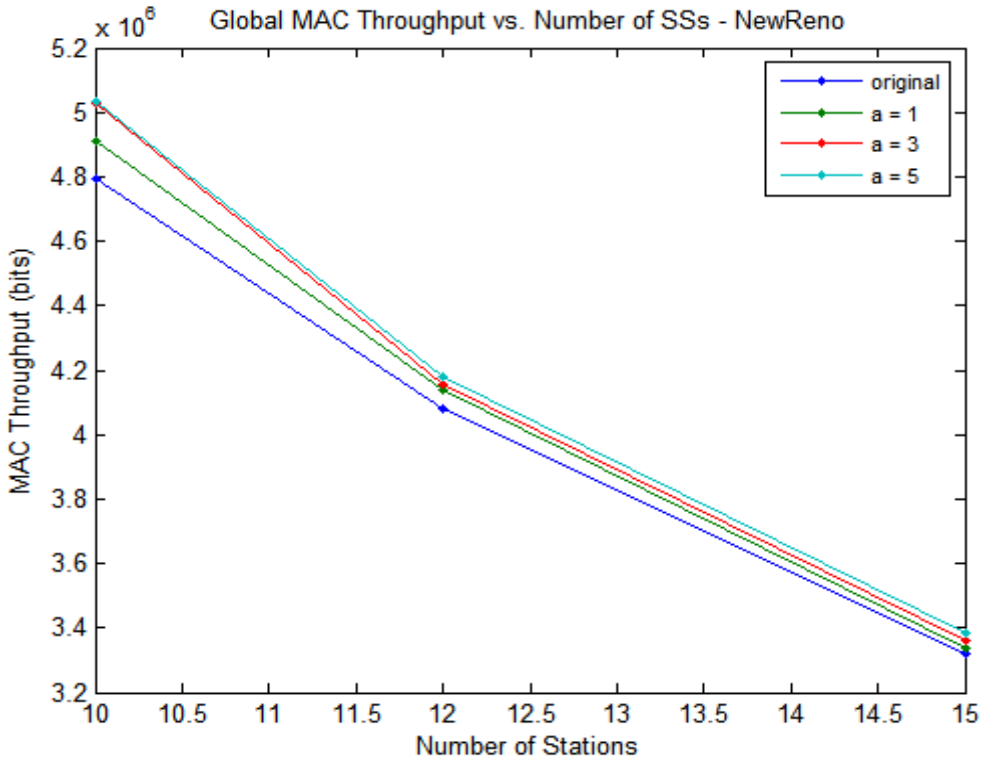

Figure 5.55: MAC throughput vs. number of station, utilizing New Reno (Zoom In)

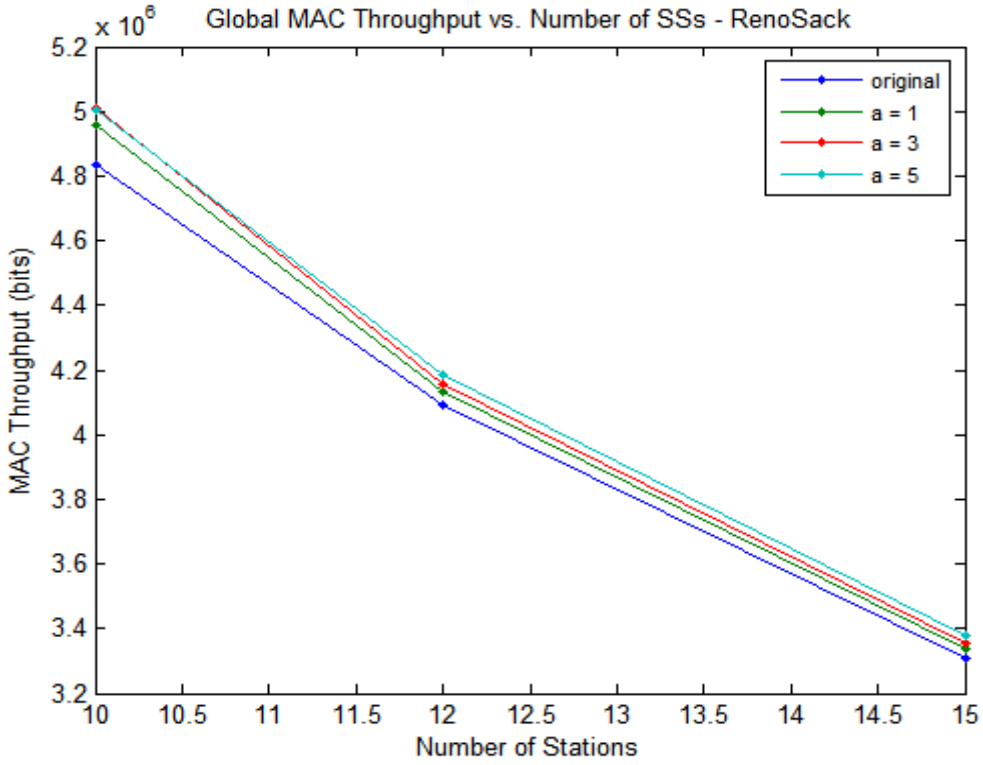

Figure 5.56: MAC throughput vs. number of station, utilizing Reno and SACK (Zoom In)

In both New Reno and Reno-SACK scenarios, the *a*=3 and *a*=5 designs are more distinguishable when *N* is twelve and fifteen. This observation demonstrates that large coefficients require larger *N*. Note that the difference between the throughput and service rate of a queue is that the service rate counts packets that are sent, but may finally be dropped by the physical link, whereas the throughput does not. Therefore, the plots of the throughput and queue service rate are very similar in a fair physical channel condition.

5.6 Base Station Analysis

The other interesting statistics to observe are the MAP-related statistics in the BS. A DL-MAP is embedded as a portion of the DL subframe (Section

2.2.1.3), and a MAP indicates the number of data bursts and the boundaries of each data burst located in the frame. A DL data burst contains data dedicated from the BS to a SS. The statistic of the number of data bursts in a DL subframe is captured, and it is plotted against various numbers of *N* as illustrated in Figure 5.57. Due to the high similarity of the graphs utilizing different TCP flavours, only the New Reno graphs are chosen as the representative in this section.

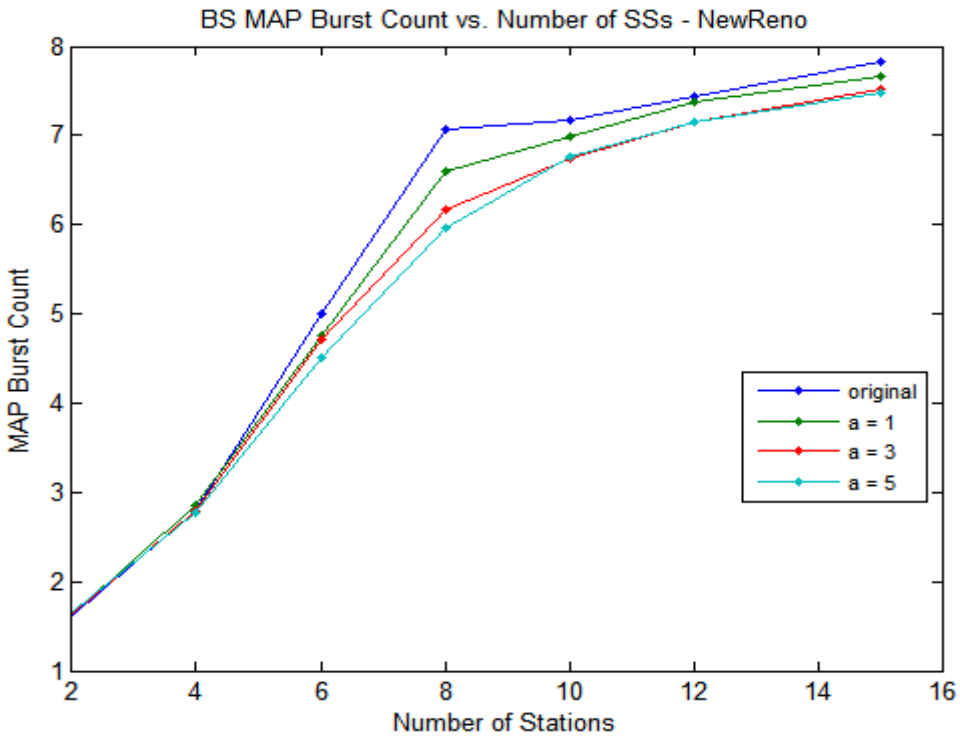

Figure 5.57: Number of burst count of the DL-MAP, utilizing New Reno

As the number of stations in the network increases, the DL subframe is divided into more portions in order to deliver data to each station. As a result, the number of burst count increases with the number of stations. Due to the additional weight granted to each queue in the proposed scheme, the partitioning of a MAC frame is not as obvious in the proposed designs as in the original. As

observed in the plots of data burst count, the numbers of data bursts in a DL subframe of the proposed designs are less than the original. However, as the number of stations in the network grows larger (i.e. N>8), the increase in the number of data bursts slows down. This phenomenon implies that the dequeuing service provided to each station is approaching the minimum QoS specification of the station. At the same time, a slow increase in burst count also suggests the BS is slowly approaching its scheduling capacity, such that it cannot accommodate more data bursts even though the number of clients has increased.

Based on the increasing trend observed in the graph of data burst count, the size of each data burst should exhibit a decreasing trend since the capacity of the BS is fixed. The size of each data burst with respect to varying numbers of stations is shown in Figure 5.58.

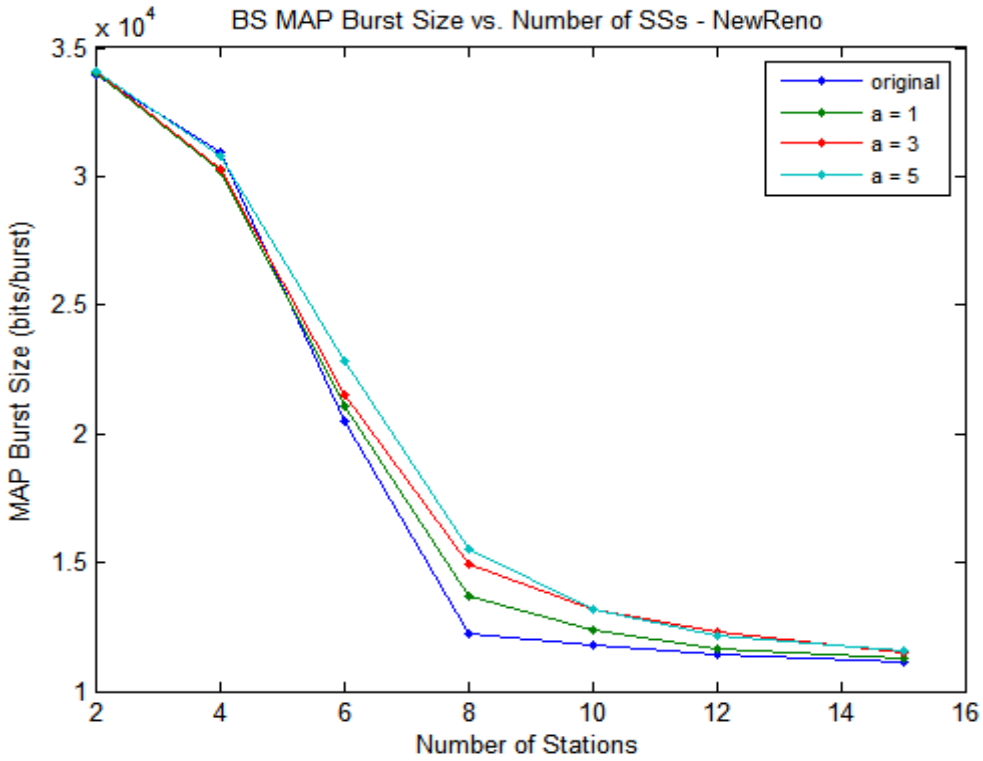

Figure 5.58: Size of each data burst in the DL-MAP, utilizing New Reno

The plots of the data burst size confirm that the size of each data burst decreases with respect to *N*. With a fixed frame capacity, the burst count and size have an inverse relationship. However, due to the additional weight granted to each queue in the proposed scheme, the sizes of the data bursts are relatively higher in the proposed designs than in the original. When *N* is greater than eight, the size of each data burst approaches a steady state. This observation confirms that the amount of data dequeued at each station is approaching its maximum allowed limit in one round (i.e. W_n').

In addition to the burst profile, the utilization of the downlink subframe is analyzed. The utilization is measured in percentage, which is the percentage of a DL subframe that is occupied by DL data bursts. The DL data burst usages of a DL subframe in the New Reno scenarios are illustrated in Figure 5.59. Since the DL-MAP and the UL-MAP together occupied as a part of the downlink subframe, the percentage of a DL subframe that is allocated to the MAP usage is also shown in Figure 5.60.

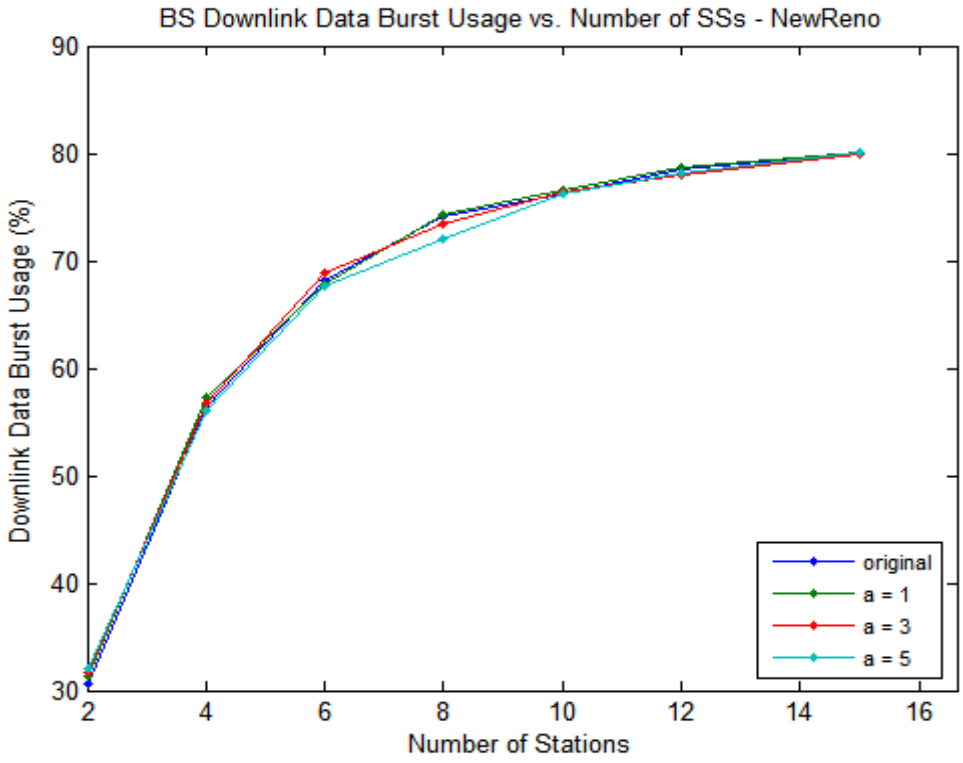

Figure 5.59: DL Data burst usage of a DL subframe, utilizing New Reno

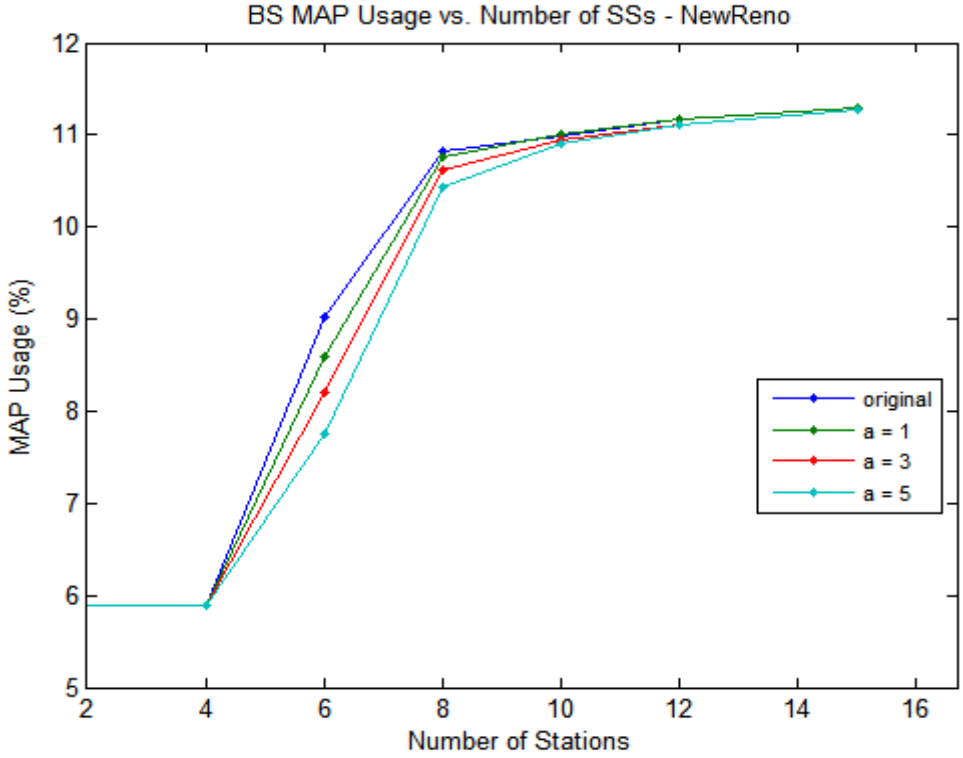

Figure 5.60: MAP usage of a DL subframe, utilizing New Reno

The utilization of the data burst in the DL subframe is indistinguishable between the proposed and original designs because the scheduling algorithm of the BS is greedy, and always attempts to accommodate as much data as possible. This results in competitive utilization between the two algorithms. Nevertheless, the occupancies of both data bursts and MAP in the DL subframe increase as the size of the network expands. The increasing utilization of the DL subframe is an indication of increasing traffic load in the network.

However, the growth of utilization in both data bursts and MAP slows down when the number of stations in the network exceeds eight. This observation confirms that the scheduling resources of the BS are approaching being fully utilized. More specifically, the combined percentage-occupancy of the data burst and MAP is approximately 91% of the DL subframe when *N* is fifteen. The observation of the network traffic load approaching the capacity of BS also explains the improvement of the proposed designs when *N* is fifteen is not as distinctive as when *N* is eight. When the traffic load of a network is catching up with the system capacity, the utilization of the network is fully exploited under most scheduling strategies. Consequently, an improvement in the system becomes more difficult to realize, as demonstrated in the plots of Section 5.5.

5.7 Weight Variations across Stations

The proposed design is an algorithm that attempts to differentiate resource allocation of the BS, based on the network congestion condition perceived at TCP. In particular, the ratio of the *cwnd* values of a station compared to that of other stations is utilized. Despite the equally fair channel

condition configured at each station in the simulation, the proposed algorithm still demonstrates an improvement over the original design on the average performance. The reason that differentiates the proposed algorithm from the original is that the physical channels are equally fair but still subject to random packet drops. The *cwnd* values thus vary randomly across stations, resulting in differentiations in the network condition assessments at a given instance. Since the *cwnd* values vary randomly with respect to time and across stations, the queue weight calculated for each queue fluctuates at different time instance as demonstrated in Figure 5.61.

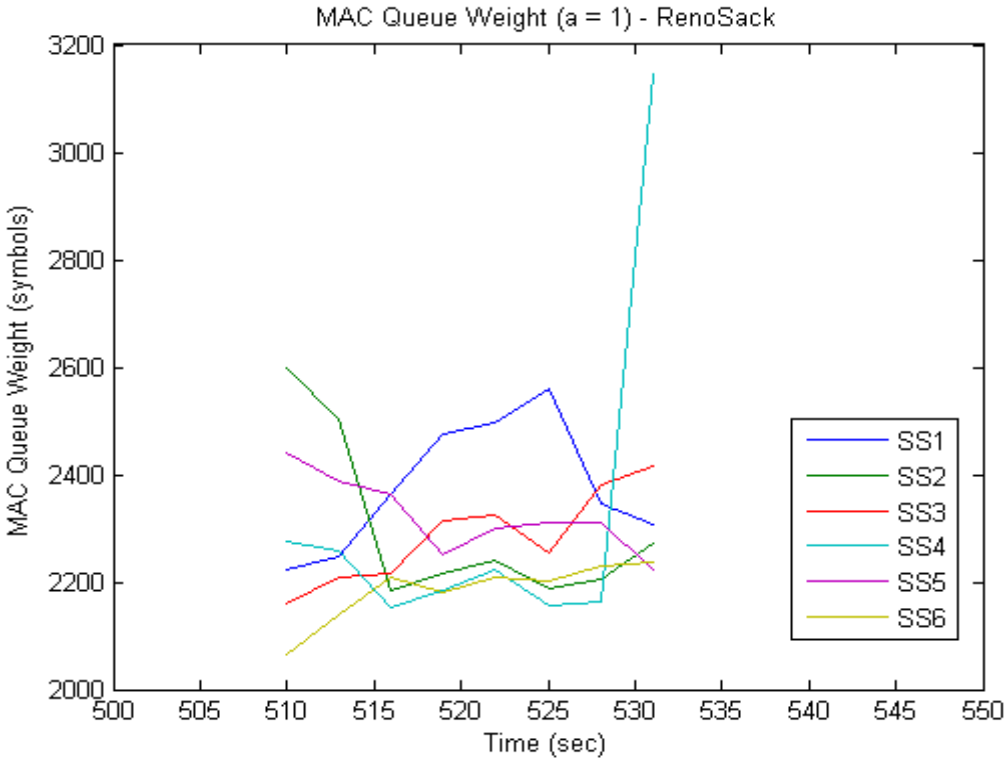

Figure 5.61: MAC queue weights of Station1 to Station6 of the 8SS-scenario, utilizing Reno-SACK combination and a=1

As a result, a queue can experience a temporal high or low service rate, depending on the *cwnd* ratio of the queue at the time. This innate adaptive

characteristic of the proposed algorithm allows it to exploit the gain of *queue diversity*, thus achieving a more efficient utilization of the bandwidth. Note that the MAP usage presented at the end of last section (i.e. Figure 5.60) indicates that when *N* is between six and twelve, the MAP usage of the proposed algorithm is less than the original. However, the average performance of the proposed designs is better than the original when *N* is between six and twelve. This indicates that the system performance is improved due to efficient utilization of the bandwidth.

The gain of queue diversity is a similar concept to the multiuser diversity gain [24] in a multiple access network. The multiuser diversity gain is an elevation in user data rates due to an adaptive scheduling algorithm, based on the physical channel states of the users. Similarly, Figure 5.61 demonstrates the weight granted to each queue is adaptive to the network congestion state over the entire transmission path.

Furthermore, the extra bandwidth granted to each queue in the proposed design can be considered as an attempt to extract more capacity out of the BS when possible. As long as the BS can sustain the capacity, all queues perform better, thus a better global performance. In other words, the proposed algorithm also exploits the system capacity in order to deliver better performance when the traffic load of the network is moderate to high. However, if the traffic load of the network approaches to the system capacity, the effect of the proposed design diminishes, as explained in the last section (Section 5.6).

CHAPTER 6: A SUMMARY AND FUTURE EXTENSIONS

The goal of this research effort is to explore the possibility of cross-layer optimization in the context of wireless networks. In this thesis, I have proposed a new cross-layer technique that interconnects the transport and MAC layers in a WMAN. In particular, the scheduler of WiMAX MAC is made aware of the congestion condition perceived at TCP. Through the knowledge of *cwnd* values provided by TCP, the MAC scheduler is capable of adaptively distributing its resources to desirable connections. The proposed algorithm is greedy since it utilizes more capacity of the BS when possible. At the same time, the proposed scheme exploits the gain of queue diversity, thus improving the transmission efficiency to attain a better average performance. At the same time, the proposed algorithm is still a bounded and stable system.

Analytical models were developed to understand the behavioural dynamics of the proposed model. In particular, the queue service rate (Equation 3.6-3.8) and expected queue delay (Equation 3.34) were developed, with particular focus on the case of moderate to high traffic intensity (i.e. case (b)). The analytical models indicate that the gain in both service rate and queue delay of the proposed scheme grows with respect to increasing N and a, but the gain is limited at a certain bound. The analytical models were further developed to analyze the effect of the MDRR queue service discipline on the RTT (Equation 3.42) of a packet. With an expression for RTT, a simple service-rate dependent

arrival rate at a MAC queue (i.e. a primitive TCP send rate model) is derived in Equation 3.46. Consequently, an arrival-rate dependent MAC queue delay is derived in Equation 3.47. Finally, by employing the analytical work published in [22], a more complete TCP send rate model, which incorporates the service rate at the MAC layer (Equation 3.60 to 3.62), is developed.

To complement the analytical results, the proposed scheme is implemented and simulated in OPNET Modeler. The simulation results indicate improvements in the end-to-end delay, file download time and throughput. The improvement becomes more evident when traffic intensity is moderate to high (i.e. large *N*), which complies with the analytical model developed. However, when the traffic load is approaching the BS capacity, the improvement is not as observable due to overly large values of *N* and full utilization of the system capacity.

Moreover, an increase in numerical value of the weight-adjusting coefficient (i.e. *a*) further emphasizes the resource allocation, based on the *cwnd* ratios. The simulation confirms that an increased coefficient improves the average performance of the network but only to an extent. The percentage gain in performance measures decreases with respect to increasing coefficients. However, both analytical models and simulation results suggest that a large coefficient value is more suitable for a large network (i.e. large *N*). Nevertheless, large coefficient should be employed with care to avoid an unstable system, and an unfair resource allocation.

Another contribution of this thesis is the development of the *cwnd*-dependent MDRR scheduler of the OPNET WiMAX MAC model. Some of the most challenging tasks were the understandings of the WiMAX standard and the implemented architecture of the OPNET WiMAX model. Another challenge of the thesis was the export of simulation raw data from OPNET for further data manipulation.

The application of the proposed algorithm is suitable for a network with diverse channel conditions, and moderate to busy traffic intensity. According to the analysis of TCP sending rate in Section 3.4.4, the proposed scheme is beneficial to TCP sending rate in a network of small RTTs. Therefore, a WMAN such as WiMAX is a network of suitable scale for the scheme. In addition, the proposed scheduler attempts to dequeue more data out of a queue in one visit, so the algorithm is ideal for a network with relatively long *walk time* between the queues. More specifically, the proposed scheduling algorithm may be more suitable in UL.

As a result, one of the possible future extensions of the current work is to implement the proposed design in the UL. Another possible extension is to develop a *credit system*, such that a station with temporary unfavourable transmission condition (i.e. low *cwnd*) is allowed to accumulate *credits* for bandwidth usage. Upon the recovery of the channel condition, the station is allowed to use the credit built up, and enjoys a temporary boost in bandwidth allocation to compensate the loss beforehand.

The other possibility is to incorporate HARQ at the MAC scheduler. The HARQ mechanism should enhance the transmission quality at the link layer. In addition, HARQ provides further information on the condition of the physical link. Finally, another possible proposal of future work is to study the distribution of the stack of *cwnd* values residing in the packets of a queue. The collection of *cwnd* values provides information on the trend of traffic variation of a queue.

APPENDICES

Appendix A: Implementation Steps and Codes Regarding the OPNET TCP Model

tcp_manager_v3 process model:

1. Declare/Invoke corresponding child process:
   a) Declare “tcp_conn_v3_sc3” as the child process
   b) Invoke child process “tcp_conn_v3_sc3”
      OPEN Enter Execs:
      ```
      op_pro_create ("tcp_conn_v3_sc3", &tcp_ptc_mem);
      ```
2. Declare/Create new Packet Format:
   a) Declare “TCP_seg_v2_sc1” as the Packet Format
   b) Create Packet Format “TCP_seg_v2_sc1”
      Function Block:
      ```
      op_pk_create_fmt ("tcp_seg_v2_sc1");
      ```
3. Create new attribute, CWND Option, for the process model
   a) Add new attribute named “CWND Option” under process model attributes interface.
   b) CWND Option attribute is set to toggle type, where only enabled and disabled options are available.
   c) Include the custom header file “tcp_v3_sc3.h” to incorporate the CWND Option field in *TcpT_Conn_Parameters*, and *TcpT_Ptc_Mem* structure. New structure names are *TcpT_Conn_Parameters_SC* and *TcpT_Ptc_Mem_SC*.
   d) Change declaration of old structure names to the new ones.
      TcpT_Conn_Parameters_SC: 1 in State Variable block, 1 in Function Block
      TcpT_Ptc_Mem_SC: 1 in State Variable block
   e) Retrieve setting from the node model and store it, Function Block *tcp_mgr_tcp_param_parse()*.

tcp_conn_v3 process model:

1. Declare Packet Format:
   a) Declare “TCP_seg_v2_sc1” as the Packet Format
   b) Create Packet Format “TCP_seg_v2_sc1”
      Function Block:
      ```
      op_pk_create_fmt ("tcp_seg_v2_sc1");
      ```
2. Change to process model to recognize the existence of CWND Option field.

a) Include the custom header file “tcp_v3_sc3.h” to incorporate the CWND Option field in *TcpT_Conn_Parameters*, and *TcpT_Ptc_Mem* structure. New structure names are *TcpT_Conn_Parameters_SC* and *TcpT_Ptc_Mem_SC*.
b) Change declaration of old structure names to the new ones.
TcpT_Conn_Parameters_SC: 1 in State Variable block
TcpT_Ptc_Mem_SC: 1 in Temporary Variable block, 1 in *init* state Enter Executives
c) Declare cwnd_enabled in State Variable block and retrieve the value from ptc_mem (parent-to-child memory) in Function Block *TCP_conn_sv_init()*.

3. Set cwnd size to the CWND Option field
a) In function *tcp_seg_send*:

```
if (cwnd_enabled)
        {
        if (op_pk_nfd_set (seg_ptr, "CWND Option", cwnd) ==
        OPC_COMPCODE_FAILURE)
        tcp_conn_error ("Unable to set CWND Option in TCP
        segment.", OPC_NIL, OPC_NIL);
        }
```

b) In function *tcp_seg_receive* checking if the value that was set in the CWND Option field

```
if (op_prg_odb_ltrace_active ("tcp _cwnd"))
        {
        Char            msg [156];
        TcpT_Size       cwnd_option;

        if (op_pk_nfd_is_set (seg_ptr, "CWND Option") ==  OPC_TRUE)
                {
                if (op_pk_nfd_get (seg_ptr, "CWND Option",
                &cwnd_option) == OPC_COMPCODE_FAILURE)
                        tcp_conn_error ("Unable to get CWND Option
                        from received TCP segment.", OPC_NIL,
                OPC_NIL);

                sprintf (msg, "Successfully retrieve CWND Option
        field [%d].", cwnd_option);
                }
        else
                {
                sprintf (msg, "CWND Option field is not set or an error
        occurs");
                }
        op_prg_odb_print_minor (msg, OPC_NIL);
        }
```

tcp_v3_sc3.h* header file*:

1. New structure is introduced to reflect CWND Option field is being added as a new TCP parameter.

```
typedef struct
	{…
	Boolean		cwnd_option_flag;
	Boolean		satisfaction_option_flag;
	} TcpT_Conn_Parameters_SC;

typedef struct
	{…
	//TcpT_Conn_Parameters*		tcp_conn_params_ptr;
	TcpT_Conn_Parameters_SC*	tcp_conn_params_ptr;
	} TcpT_Ptc_Mem_SC;
```

Appendix B: Implementations of Extraction, Storage and Removal of *Cwnd*

Extraction of *cwnd* values: (*wimax_mac_sc3_wt_adjustable* process model)

- The *cwnd* value is extracted from the data packet arriving at the MAC layer, in the WiMAX MAC root process.

```
static void
wimax_support_mac_pk_in_queue_efficiency_off_sc (Packet* pkptr, WimaxT_Shaper_Queue_Elem* sq_elem_ptr, WimaxT_Service_Flow* service_flow_ptr)
{
……
/* We generate a BWR only when its computed size is positive.        */
if (bwr_size_bits > 0)
        {
        /* Susan-code: Retreive CWND size and ready to pass it down to the
        child process */
        cwnd_size = wimax_mac_cwnd_get (pkptr, &frag_status);

        /* Create and insert a new BWR into the bandwidth request    */
        /* queue.                                                    */
        wimax_support_bw_request_insert_sc (sq_elem_ptr, (int) bwr_size_bits,
        cwnd_size, frag_status);
        }
}

/* Susan-code: This fuction is to extract the CWND Option field in a TCP packet   */
int
wimax_mac_cwnd_get (Packet* pkptr, IpT_Pkt_Frag_Info* frag_status_ptr)
{
int                                     cwnd;
FIN (wimax_mac_cwnd_get (pkptr));

/* Check if the ARP sim efficiency is disabled. In that case, the IP packet does   */
/* not contain the necessary IP socket information. ARP cannot function            */
/* correctly in Wimax, because broadcast is not supported. Warn the user.          */
```

```
if (OPC_TRUE == ip_arp_sim_eff_sim_attr_get (OPC_FALSE))
	{
	/* Susan-code: Extract the CWND Option field from TCP header.   */
	ip_support_ip_pkt_cwnd_extract (pkptr, &cwnd, frag_status_ptr);
	}
else
	{
	/* When PHY pipelines are used, layer 2 broadcast is also  */
	/* available. In that case, refrain from ARP warning.          */
	if (global_efficiency_less_than_phy)
		{
		/* Warn the user to turn the ARP Sim Efficiency on. */
		wimax_arp_efficiency_disabled_log_write ();
		}
	cwnd = -1;
	}
FRET (cwnd);
}

DLLEXPORT Compcode
ip_support_ip_pkt_cwnd_extract (Packet* pkptr, int* cwnd_ptr, IpT_Pkt_Frag_Info* frag_status_ptr)
{
IpT_Dgram_Fields*		pk_fd_ptr = OPC_NIL;
Boolean				cwnd_info_read = OPC_FALSE;
List*				parent_pkt_lptr;
Packet*				pk_data_ptr = OPC_NIL, *seg_pkptr;
FIN (ip_support_ip_pkt_cwnd_extract (pkptr));

/* Value initialize to invalid values so that we eliminate multiple	*/
*cwnd_ptr = -1;
*frag_status_ptr = IpC_Pkt_Frag_Not_Available;

/* Get the content of the IP header fields */
if (op_pk_nfd_access (pkptr, "fields", &pk_fd_ptr) == OPC_COMPCODE_FAILURE)
```

```
        {
        }
else
        {
        /* For non-tracer packets, socket information is present in the regular location. */
        /* Look at the protocol field and *not* the encap flag. If this is a tracer packet   */
        /* that is being tunneled through another IP packet, we must  return the socket*/
        /* information of the outer pkt and *not* the socket information of the tracer.    */
        if (pk_fd_ptr->protocol != IpC_Protocol_Basetraf)
                {
                /* Port information can be obtained from packets that are not         */
                /* fragments, or from initial fragments.                              */
                if (!pk_fd_ptr->frag || (pk_fd_ptr->offset == 0))
                        {
                        if (op_pk_nfd_is_set (pkptr, "data"))
                                {
                                /* Get the data from the IP-datagram.          */
                                op_pk_nfd_get (pkptr, "data", &pk_data_ptr);
                                /* For fragments, this data is a segment, not the original*/
                                /* transport payload.                                 */
                                if (pk_fd_ptr->frag)
                                        {
                                        seg_pkptr = pk_data_ptr;
                                        parent_pkt_lptr =
                                        op_sar_seg_parent_packets_access (seg_pkptr);
                                        pk_data_ptr = (Packet *) op_prg_list_remove
                                        (parent_pkt_lptr, OPC_LISTPOS_HEAD);
                                        /* Parent list should be empty because IP segments
                                        are not */
                                        /* comprised of multiple packets.             */
                                        op_prg_mem_free (parent_pkt_lptr);

                                        /* Susan-code */
                                        *frag_status_ptr = IpC_Pkt_Frag_Initial;
                                        }
                                else
```

```
                    {
                    *frag_status_ptr = IpC_Pkt_Not_Frag;
                    }
                /* Check if the transport protocol is TCP       */
                /* to acquire the TCP  port number.                    */
                if (pk_fd_ptr->protocol == IpC_Protocol_Tcp)
                    {
                    /* Susan-code */
                    // Get the CWND Option field from the TCP header
                   if (op_pk_nfd_is_set (pk_data_ptr, "CWND Option"))
                        {
                        op_pk_nfd_access (pk_data_ptr, "CWND
                            Option", cwnd_ptr);
                        }
                    }
                /* If this is a fragment, then destroy the copy transport */
                /* payload.                                            */
                if (pk_fd_ptr->frag)
                    {
                    op_pk_destroy (pk_data_ptr);
                    pk_data_ptr = seg_pkptr;
                    }
                /* Set the original payload (segment or actual) payload
                back in the IP datagram.                            */
                op_pk_nfd_set (pkptr, "data", pk_data_ptr);
                }
            else
                {
                /* The data field is not set */
                }
            }
        else
            {
            /* Susan-code */
            *frag_status_ptr = IpC_Pkt_Frag_Non_Initial;
```

```
                    /* This is a fragmented packet and not the initial fragment   */
                    }
                cwnd_info_read = OPC_TRUE;
                }
        }
FRET ((cwnd_info_read) ? OPC_COMPCODE_SUCCESS : OPC_COMPCODE_FAILURE);
}
```

Storage of *cwnd* values: (*wimax_bs_control_sc3_wt_adjustable* process model)

- When the bandwidth request invokes the BS control child process, the request is enqueued in the buffer. Along with the bandwidth request, the corresponding *cwnd* value is retrieved and stored in the list structure

```
/* Susan-code */
static Boolean
wimax_bs_control_sched_bw_req_ps_insert (WimaxT_Bs_Scheduler_Handle* sched_hdlptr, WimaxT_Request_Element* bwr_ptr, int conn_id, int cwnd_value, IpT_Pkt_Frag_Info frag_info)
{
WimaxT_Polling_Service_Queue* ps_q_ptr;
int*    cwnd_value_ptr;
/** Insert a BW request into the corresponding PS queue based    **/
/** on the specified connection ID.                              **/
FIN (wimax_bs_control_sched_bw_req_ps_insert ());

ps_q_ptr = (WimaxT_Polling_Service_Queue *) wimax_bs_control_sched_poll_serv_q_access_by_conn_id (sched_hdlptr, conn_id);
/* Insert the incomming BW request.                             */
oms_buffer_enqueue (ps_q_ptr->buffer, bwr_ptr, OPC_NIL, op_sim_time ());

/* Susan-code: Insert the corresponding CWND size of each request (payload) */
cwnd_value_ptr = (int *) op_prg_mem_alloc (sizeof (int));
if (frag_info == IpC_Pkt_Not_Frag || frag_info == IpC_Pkt_Frag_Initial)
        {
        /* Insert CWND value as it is since this packet is either not fragmented,    */
        /* or is the initial fragment                                                 */
```

***cwnd_value_ptr = cwnd_value**;

ps_q_ptr->last_valid_cwnd = cwnd_value;

}

else

{

/* Insert prvious CWND value if this packet is fragmented and not the initial fragment, or the fragmentation info is not proper */

*cwnd_value_ptr = ps_q_ptr->last_valid_cwnd;

}

op_prg_list_insert (ps_q_ptr->cwnd_list_ptr, cwnd_value_ptr, OPC_LISTPOS_TAIL);

FRET (OPC_TRUE);

}

Removal of *cwnd* values: (*wimax_bs_control_sc3_wt_adjustable* process model)

- The *cwnd* values are removed when the corresponding bandwidth request is removed (dequeue) from the queue in BS control plane

static WimaxT_Request_Element*

wimax_bs_control_sched_bw_req_dequeue_from_ps (WimaxT_Bs_Scheduler_Handle* sched_ptr, int q_index)

{

WimaxT_Polling_Service_Queue* ps_q_info_ptr;

WimaxT_Request_Element* bw_req_ptr;

int* cwnd_list_value_ptr;

/** This function dequeues the next BW request available in **/

/** the PS queue indicated by the q_index. **/

FIN (wimax_bs_control_sched_bw_req_dequeue_from_buffer);

/* Access the PS queue that corresponds to q_index. */

ps_q_info_ptr = (WimaxT_Polling_Service_Queue *) wimax_bs_control_sched_poll_serv_q_access_by_index (sched_ptr, q_index);

/* Extract the next BW request from the buffer. */

bw_req_ptr = (WimaxT_Request_Element *) oms_buffer_dequeue (ps_q_info_ptr->buffer, 0, op_sim_time ());

```
/* Susan-code: toss away the CWND value of the request upon dequeueing */
cwnd_list_value_ptr = (int *) op_prg_list_remove (ps_q_info_ptr->cwnd_list_ptr, OPC_LISTPOS_HEAD);
/* Susan-trace */
if (op_prg_odb_ltrace_active ("wimax_cwnd") || op_prg_odb_ltrace_active ("cwnd"))
        {
        char    msg [256];
        sprintf (msg, "Dequeue from PS CID[%d]: [%d] bits [%d] symbols, CWND[%d] Q-weight[%d], Q-deficit counter[%d]", bw_req_ptr->conn_id, bw_req_ptr->bwr_size_bits, bw_req_ptr->bwr_size_symbols, *cwnd_list_value_ptr, ps_q_info_ptr->weight, ps_q_info_ptr->deficit_counter);
        op_prg_odb_print_minor (msg, OPC_NIL);
        }
op_prg_mem_free (cwnd_list_value_ptr);


ps_q_info_ptr->num_elems_dequeued++;
ps_q_info_ptr->num_bits_dequeued += bw_req_ptr->bwr_size_bits;
if (op_prg_odb_ltrace_active ("wimax_sched_deq") || op_prg_odb_ltrace_active ("wimax_deq_stat"))
        {
        char    msg [256];
        sprintf (msg, "The scheduler has dequeued CID-%d [%d] requests and [%d] bits, so far, in this schedulling round", ps_q_info_ptr->conn_id, ps_q_info_ptr->num_elems_dequeued, ps_q_info_ptr->num_bits_dequeued);
        op_prg_odb_print_minor (msg, OPC_NIL);
        }


FRET (bw_req_ptr);
}
```

Appendix C: Implementations of the Queue Weight Calculation and Modified MDRR Queuing Service Discipline

Queue Weight Calculation: (*wimax_bs_control_sc3_wt_adjustable* process model)

- The weight of a queue is calculated before the scheduling (dequeue) process.

```
static WimaxT_Map*
wimax_bs_control_one_ofdma_map_generate (WimaxT_Bs_Scheduler_Handle* sched_ptr, int* free_symbols_ptr, int* dl_free_symbols_ptr, int ie_size,
WimaxT_Region type, int* maps_offset_ptr, int num_perennials)
{……
/* Step 1: Take out as many elements as possible from the scheduler */
while (bwr_count > 0)
        {
        /* Susan-code: Obtain CWND values across all queues and refresh the */
        /* queue weight according to the CWND values */
        wimax_bs_control_sched_cwnd_weight_adjust (sched_ptr);

        /* Extract the request. */
        elem_ptr = (WimaxT_Request_Element *)
                wimax_bs_control_sched_bw_req_dequeue (sched_ptr);
        if (elem_ptr == OPC_NIL)
                break;
        ……
        }
……
}

/* Susan-code: Obtain CWND values across all queues and refresh the */
/* queue weight according to the CWND values */
void
wimax_bs_control_sched_cwnd_weight_adjust (WimaxT_Bs_Scheduler_Handle* sched_ptr)
{
WimaxT_Polling_Service_Queue*   ps_q_info_ptr;
```

```
int                                     ps_q_count, index = 0;
int                                     *cwnd_size_ptr;
int                                     cwnd_total = 0, last_valid_cwnd_total = 0;
int                                     original_weight = 0, adjust_weight = 0;
FIN (wimax_bs_control_sched_cwnd_weight_adjust());

/* Get the total number of queue count, and get the first queue                */
ps_q_count = wimax_bs_control_sched_poll_serv_q_count (sched_ptr);

/* 1. Determine the total of CWND values of the first request in each queue   */
for (index=0; index<ps_q_count; index++)
        {
        ps_q_info_ptr = (WimaxT_Polling_Service_Queue *)
        wimax_bs_control_sched_poll_serv_q_access_by_index (sched_ptr, index);


        /* last_valid_cwnd_total calculation                                   */
        last_valid_cwnd_total += ps_q_info_ptr->last_valid_cwnd;
        /* cwnd_total calculation                                              */
        if (op_prg_list_size(ps_q_info_ptr->cwnd_list_ptr) > 0)
                {
                cwnd_size_ptr = (int *) op_prg_list_access (ps_q_info_ptr
                ->cwnd_list_ptr, OPC_LISTPOS_HEAD);
                cwnd_total += *cwnd_size_ptr;
                /* Record the cwnd size statistics      */
                wimax_bs_control_sched_cwnd_stat_update (ps_q_info_ptr,
                *cwnd_size_ptr);
                if (op_prg_odb_ltrace_active ("wimax_cwnd") ||
                op_prg_odb_ltrace_active ("cwnd"))
                        {
                        char    msg [256];
                        sprintf (msg, "1st request of Q-index[%d] CID[%d} has a last-valid
                        CWND[%d] and CWND [%d]", index, ps_q_info_ptr->conn_id,
                        ps_q_info_ptr->last_valid_cwnd, *cwnd_size_ptr);
                        op_prg_odb_print_minor (msg, OPC_NIL);
                        }
                }
```

```
	else
		{
		if (op_prg_odb_ltrace_active ("wimax_cwnd") || op_prg_odb_ltrace_active ("cwnd"))
			{
			char	msg [256];
			sprintf (msg, "\tQ-index[%d] CID[%d] is empty and has last-valid CWND[%d]", index, ps_q_info_ptr->conn_id, ps_q_info_ptr->last_valid_cwnd);
			op_prg_odb_print_minor (msg, OPC_NIL);
			}
		}
	}
//check for total CWND value. If 0, set to 1
if (cwnd_total == 0)
	{
	cwnd_total = 1;
	if (op_prg_odb_ltrace_active ("wimax_cwnd") || op_prg_odb_ltrace_active ("cwnd"))
		{
		printf ("\nTime[%.6f]: Total CWND value is 0, and is reset to 1.  Total Q-count is [%d]\n", op_sim_time (),ps_q_count);
		}
	}
if (last_valid_cwnd_total == 0)
	{
	last_valid_cwnd_total = 1;
	if (op_prg_odb_ltrace_active ("wimax_cwnd") || op_prg_odb_ltrace_active ("cwnd"))
		{
		printf ("\nTime[%.6f]: Total last-valid-CWND value is 0, and is reset to 1. Total Q-count is [%d]\n", op_sim_time (),ps_q_count);
		}
	}
```

/* 2. Adjust the Q-weight according to the CWND value proportion */

```
for (index=0; index<ps_q_count; index++)
```

```
{
ps_q_info_ptr = (WimaxT_Polling_Service_Queue *) wimax_bs_control_sched_poll_serv_q_access_by_index (sched_ptr, index);
original_weight = ps_q_info_ptr->original_weight;
/* cwnd-adjusted weight calculation */
if (op_prg_list_size(ps_q_info_ptr->cwnd_list_ptr) > 0)
        {
        cwnd_size_ptr = (int *) op_prg_list_access (ps_q_info_ptr->cwnd_list_ptr, OPC_LISTPOS_HEAD);
        adjust_weight = original_weight * (*cwnd_size_ptr) / cwnd_total;
        }
else
        adjust_weight = 0;

ps_q_info_ptr->cwnd_weight = adjust_weight;
//ps_q_info_ptr->weight = original_weight + (1*adjust_weight);
//ps_q_info_ptr->last_valid_cwnd_weight = 1*(original_weight * ps_q_info_ptr->last_valid_cwnd / last_valid_cwnd_total);
ps_q_info_ptr->weight = original_weight + (weight_adjust_factor * adjust_weight);
ps_q_info_ptr->last_valid_cwnd_weight = weight_adjust_factor*(original_weight * ps_q_info_ptr->last_valid_cwnd / last_valid_cwnd_total);

if (op_prg_odb_ltrace_active ("wimax_cwnd") || op_prg_odb_ltrace_active ("cwnd"))
        {
        char    msg1 [256];
        char    msg2 [256];
        sprintf (msg1, "Weight adjust: Q[%d] CID[%d]: original weight[%d] + weight_adjust_factor[%f] * cwnd-adjusted weight[%d] = final weight [%d]",
        index, ps_q_info_ptr->conn_id, ps_q_info_ptr->original_weight, weight_adjust_factor, ps_q_info_ptr->cwnd_weight, ps_q_info_ptr->weight);
        sprintf (msg2, "\tdeficit couter[%d], original weight[%d] + last_valid_cwnd adjustweight[%d] = last-valid weight [%d]", ps_q_info_ptr->deficit_counter, original_weight, ps_q_info_ptr->last_valid_cwnd_weight, original_weight + ps_q_info_ptr->last_valid_cwnd_weight);
        op_prg_odb_print_minor (msg1, msg2, OPC_NIL);
```

```
                }
        }
FOUT;
}
```

Modified MDRR Queuing Service Discipline: (*wimax_bs_control_sc3_wt_adjustable* process model)

- This function implements the flowchart of Figure 3.2

```
static int
wimax_bs_control_sched_mdrr_q_select (WimaxT_Bs_Scheduler_Handle* sched_ptr, WimaxT_Requests_Queue_Handle* q_hdl_ptr, Boolean force_block_current_queue)
{
int                                 num_queues;
int                                 q_in_service;
WimaxT_Polling_Service_Queue*       ps_q_info_ptr;
Boolean                             ok_to_service;
int                                 last_valid_weight = 0;
/** Based on the MDRR scheduling algorithm, select the next BW request          **/
/** queue to be served. MDRR is applied only for rtPS and nrtPS services.       **/
FIN (wimax_bs_control_sched_mdrr_q_select ());

/* If none of the queues were in service earlier, start with first queue.      */
q_in_service = (q_hdl_ptr->current_q_idx == WIMAXC_Q_INDEX_NONE) ? WIMAXC_Q_INDEX_FIRST : q_hdl_ptr->current_q_idx ;
ps_q_info_ptr = (WimaxT_Polling_Service_Queue *) prg_list_access (q_hdl_ptr->polling_service_q_lptr, q_in_service);
/* Get the number of queues.                                                   */
num_queues  =  prg_list_size (q_hdl_ptr->polling_service_q_lptr);

/* Service a queue until deficit becomes negative or until the queue becomes empty. */
if ((ps_q_info_ptr->deficit_counter > 0) && (!oms_buffer_is_empty (ps_q_info_ptr->buffer)) && !force_block_current_queue)
        {
        q_hdl_ptr->current_q_idx = q_in_service;
        FRET (q_in_service);
        }
```

```
//else if (oms_buffer_is_empty (ps_q_info_ptr->buffer))
/* Susan-code: If the serving queue has only one request left, it will be      */
/* dequeued despite its deficit counter value                                  */
else if ( ((int) oms_buffer_num_elements_get (ps_q_info_ptr->buffer) == 1)
&& !force_block_current_queue)
        {
        q_hdl_ptr->current_q_idx = q_in_service;
        if (op_prg_odb_ltrace_active ("wimax_sched_deq"))
                {
                char    msg [256];
                sprintf (msg, "MDRR: current queue has only 1 request left, serve it
                anyway if space allowed");
                op_prg_odb_print_minor (msg, OPC_NIL);
                }
        FRET (q_in_service);
        }

/* If there are no more packets in any of the PS buffers, quit.               */
if (wimax_bs_control_sched_all_ps_queues_num_bwr_get (sched_ptr) <= 0)
        {
        FRET (WIMAXC_Q_INDEX_NONE);
        }

/* Loop to find next candidate queue for service.                             */
do
        {
        ok_to_service = OPC_TRUE;
        /* Wrap around queues if last else move to next queue.                */
        q_in_service = ((q_in_service == num_queues - 1) ? 0 : q_in_service + 1);

        /* Susan-code: Keep track of the number of times that the scheduler has  */
        /* wrapped through the last of the queues                             */
        if (q_in_service == 0)
                q_hdl_ptr->wrap_around_counter++;
```

```
/* Get handle to new queue */
ps_q_info_ptr = (WimaxT_Polling_Service_Queue *) prg_list_access (q_hdl_ptr->polling_service_q_lptr, q_in_service);

/* Check if there are pending requests in this buffer.                    */
if (oms_buffer_is_empty (ps_q_info_ptr->buffer))
        {
        /* AKP: If in debt add weight */
        //if (ps_q_info_ptr->deficit_counter < ps_q_info_ptr->weight)
        //        ps_q_info_ptr->deficit_counter += ps_q_info_ptr->weight;
        last_valid_weight = ps_q_info_ptr->original_weight + ps_q_info_ptr->last_valid_cwnd_weight;
        if (ps_q_info_ptr->deficit_counter < last_valid_weight)
                {
                ps_q_info_ptr->deficit_counter += last_valid_weight;

                if (op_prg_odb_ltrace_active ("wimax_sched_deq"))
                        {
                        printf (" - queue(%d) (CID-%d) is empty, deficit_counter[%d] restores with last_valid_weight to [%d].\n", q_in_service, ps_q_info_ptr->conn_id, ps_q_info_ptr->deficit_counter - last_valid_weight, ps_q_info_ptr->deficit_counter);
                        }
                /* Susan-code: update weight statistic          */
                wimax_bs_control_sched_weight_stat_update (ps_q_info_ptr, last_valid_weight);
                }

        if (op_prg_odb_ltrace_active ("wimax_sched_deq"))
                {
                printf (" - queue(%d) empty => skip it.\n", q_in_service);
                }

        /* Ignore empty queue.                                            */
        ok_to_service = OPC_FALSE;
        }
```

```
else
        {
        /* Check the deficit counter of this queue.                    */
        if (ps_q_info_ptr->deficit_counter <=0)
                {
                ps_q_info_ptr->deficit_counter += ps_q_info_ptr->weight;


                if (op_prg_odb_ltrace_active ("wimax_sched_deq"))
                        {
                        printf (" - queue(%d) (CID-%d) is NOT empty,
                        deficit_counter[%d] restores with weight to [%d].\n",
                        q_in_service, ps_q_info_ptr->conn_id, ps_q_info_ptr-
                        >deficit_counter - ps_q_info_ptr->weight, ps_q_info_ptr-
                        >deficit_counter);
                        }
                /* Susan-code: update weight statistic          */
                wimax_bs_control_sched_weight_stat_update (ps_q_info_ptr,
                ps_q_info_ptr->weight);
                }
        /* If deficit counter is still negative skip this queue.       */
        if (ps_q_info_ptr->deficit_counter <=0)
                ok_to_service = OPC_FALSE;


        if (op_prg_odb_ltrace_active ("wimax_sched_deq"))
                {
                if (ok_to_service)
                        printf (" => service it.\n");
                else
                        printf (" => skip it. \n");
                }
        }
    /* Until there is a queue to service. There id at least one BW request     */
    /* so this condition must be true at some time.                            */
    } while (!ok_to_service);


/* Return the index of the queue in service.                                   */
```

```
q_hdl_ptr->current_q_idx = q_in_service;
FRET (q_in_service);
}
```

REFERENCE LIST

[1] B. Sardar and D. Saha, "A survey of TCP enhancements for last-hop wireless networks," *IEEE Commun. Surveys Tuts.*, vol. 8, pp. 20-34, 3rd Qtr. 2006.

[2] V. Srivastava and M. Motani, "Cross-layer design: a survey and the road ahead," *IEEE Commun. Mag.,* vol. 43, pp. 112-119, Dec. 2005.

[3] F. Foukalas, V. Gazis and N. Alonistioti, "Cross-layer design proposals for wireless mobile networks: a survey and taxonomy," *IEEE Commun. Surveys Tuts.,* vol. 10, pp. 70-85, First Quarter 2008.

[4] G. Song and Y. Li, "Utility-based resource allocation and scheduling in OFDM-based wireless broadband networks," *IEEE Commun. Mag.,* vol. 43, pp. 127-134, Dec. 2005.

[5] S. Toumpis and A. J. Goldsmith, "Performance, optimization, and cross-layer design of media access protocols for wireless ad hoc networks," in *Communications, 2003. ICC '03. IEEE International Conference on,* 11-15 May 2003, pp. 2234-2240.

[6] K. Ramakrishnan and S. Floyd, “A proposal to add explicit congestion notification (ECN) to IP,” RFC 2481, Jan. 1999.

[7] D. Kliazovich and F. Graneill, "A cross-layer scheme for TCP performance improvement in wireless LANs," in *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE,* Dec. 2004, pp. 840-844.

[8] E. Park, D. Kim, H. Kim and C. Choi, "A cross-layer approach for per-station fairness in TCP over WLANs," *IEEE Trans. Mobile Comput.,* vol. 7, pp. 898-911, July 2008.

[9] G. Giambene, *Resource Management in Satellite Networks, Optimization and Cross-Layer Design.* New York: Springer Science, 2007, Chapter 9.

[10] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach Featuring the Internet.* ,2nd ed. Boston: Addison Wesley, 2003, pp. 752.

[11] W. R. Stevens, *TCP/IP llustrated: The Protocols.* , vol. 1, Boston: Addison Wesley, 1994, pp. 576.

[12] M. Mathis, J. Mahdavi, S. Floyd and A. Romanow. (1996, October). TCP selective acknowledgment options. [Online]. Available: http://www.ietf.org/rfc/rfc2018.txt

[13] A. Ghosh, D. R. Wolter, J. G. Andrews and R. Chen, "Broadband wireless access with WiMax/802.16: current performance benchmarks and future potential," *IEEE Commun. Mag.,* vol. 43, pp. 129-136, Feb 2005.

[14] A. Yarali and S. Rahman, "WiMAX broadband wireless access technology: Services, architecture and deployment models," in *Electrical and Computer Engineering, 2008. CCECE 2008. Canadian Conference on,* 4-7 May 2008, pp. 77-82.

[15] B. Li, Y. Qin, C. P. Low and C. L. Gwee, "A survey on mobile WiMAX ," *IEEE Commun. Mag.,* vol. 45, pp. 70-75, December 2007.

[16] OPNET Technologies, Inc., "Introduction to WiMAX," presented at the Technology Tutorials Session 1827 OPNETWORK 2007, Washington, DC*,* Aug. 2007.

[17] IEEE 802.16 Working Group on Broadband Wireless Access, “IEEE Standard for Local and metropolitan area networks: Part 16: Air Interface for Fixed Broadband Wireless Access Systems,” IEEE Std 802.16-2004, Oct. 1, 2004

[18] IEEE 802.16 Working Group on Broadband Wireless Access, “IEEE Standard for Local and metropolitan area networks: Part 16: Air Interface for Fixed Broadband Wireless Access Systems: Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands and Corrigendum 1,” IEEE Std 802.16e-2005 and IEEE Std 802.16-2004/Cor 1-2005, Feb. 28, 2006

[19] OPNET Technologies, Inc., "Understanding WiMAX Model Internals and Interfaces,” presented at the Discrete Event Simulation for R&D Session 1571 OPNETWORK 2007, Washington, DC*,* Aug. 2007.

[20] Cisco Systems, Inc. (2004, Jan.). Understanding and Configuring MDRR/WRED on the Cisco 12000 Series Internet Router. [Online]. Available: http://www.cisco.com/warp/public/63/mdrr_wred_overview.html

[21] J. F. Hayes and T. V. J. Ganesh Babu, *Modeling and Analysis of Telecommunications Networks*. Hoboken, New Jersey: John Wiley & Sons, Inc., 2004, pp. 69.

[22] J. Padhye, V. Firoiui, D. F. Towsley and J. F. Kurose, "Modeling TCP Reno performance: a simple model and its empirical validation," *IEEE/ACM Trans. Netw.,* vol. 8, pp. 133-145, Apr. 2000.

[23] Z. Chen, T. Bu, M. Ammar and D. Towsley, "Comments on Modeling TCP Reno performance: a simple model and its empirical validation," *IEEE/ACM Trans. Netw.,* vol. 14, pp. 451-453, Apr. 2006.

[24] S. Shakkottai, T. S. Rappaport and P. C. Karlsson, "Cross-layer design for wireless networks," *IEEE Commun. Mag.,* vol. 41, pp. 74-80, Oct. 2003.

[25] A. J. Paulraj, D. A. Gore, R. U. Nabar and H. Bolcskeil, "An overview of MIMO communications - a key to gigabit wireless," *Proc. IEEE,* vol. 92, pp. 198-218, Feb. 2004.

[26] A. Cantoni and L. C. Godara, “Fast algorithm for time domain broadband adaptive array processing,” *IEEE Trans. Aerosp. Electron. Syst.*, vol. AES-18, pp. 682-699, Sept. 1982.

[27] D. Johnson and D. Dudgeon, *Array Signal Processing: Concepts and Techniques,* Prentice-Hall, Englewood Cliffs, NJ, 1993.

[28] J. C. Liberti and T. S. Rappaport, *Smart Antenna for Wireless Communications: IS-95 and Third Generation CDMA Applications,* Prentice-Hall, Englewood Cliffs, NJ, 1999.

[29] D. Branlund, “Stacked Carrier OFDM: Providing High Spectral Efficiency with Greater Coverage,” *Wireless Communications Association International 7th annual Technical Symposium,* San Jose, CA, Jan. 2001.[1]

[30] A. Demers, S. Keshav and S. Shenker, "Analysis and Simulation of a Fair Queuing Algorithm," *Internetworking: Research and Experience*, Vol. 1, No. 1, pp. 3-26, 1990

[31] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," *IEEE/ACM Trans. Netw.*, Vol. 1, No. 3, pp. 344-357, June 1993.

[32] X. Yang, M. Venkatachalam and S. Mohanty, “Exploiting the MAC layer flexibility of WiMAX to systematically enhance TCP performance,” *IEEE Mobile WiMAX Symposium, 2007*. 25-29 March 2007, pp. 60-65

[1] Contact WCAI at +1(202)452 7823 or sahar@wcai.com to obtain copy.