Buffer Management for WiMax/802.16 Subscriber Stations

by
Bing Zhang

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE

School of Computer Science

at

CARLETON UNIVERSITY

Ottawa, Ontario

September, 2007

© Copyright by Bing Zhang, 2007

Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 978-0-494-33685-4

Our file Notre référence

ISBN: 978-0-494-33685-4

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Table of Contents

List of Tables	iv
List of Figures	vi
Abstract	viii
Acknowledgements	ix
Chapter 1 Introduction	1
1.1 Introduction	1
1.1.1 WiMax/802.16 Broadband Wireless Access Systems	1
1.1.2 Traffic Model	3
1.1.3 Quality of Service	4
1.1.4 Buffer Management	5
1.2 Motivation	6
1.3 Statement of the Problem	7
1.4 Contribution and Results	7
1.5 Organization of Thesis	8
Chapter 2 Literature Review	9
2.1 Buffer Management	9
2.2 WiMax/802.16	13
Chapter 3 The Admission Control and Dynamic Buffer Allocation Scheme	15
3.1 Terminology	16
3.2 Admission Control Algorithm	16
3.3 Dynamic Buffer Allocation Algorithm	19

Chapter 4	Simulation of the Admission Control and Dynamic Buffer Allocation Scheme	31
4.1	Simulation Objective	31
4.2	Simulation Environment	31
4.3	Simulation Structure	32
4.4	Simulation Algorithms	40
4.4.1	Algorithms for Our Proposed Buffer Management Scheme . .	40
4.4.2	Algorithms for the Pushout Buffer Management Scheme . . .	54
4.5	Simulation Results	56
Chapter 5	Conclusions and Future Work	72
5.1	Conclusions	72
5.2	Future Work	73
Appendix A	Glossary	74
Bibliography		76

List of Tables

Table 2.1	Comparisons of buffer management schemes. Note: high priority packet loss rate (HPLR), overall packet loss rate (OPLR), useless real-time packet handle (URTPH), access control (AC), dynamic queue threshold (DT), hierarchical dynamic complete buffer partition (H-DCBP), hierarchical dynamic queue threshold (H-DT), no consideration (N/C), fast handover buffer management (FHBM).	13
Table 4.1	Simulation Parameters. Note: average connection interarrival (ACI), average connection duration (ACD), average burst interarrival (ABI), average burst duration (ABD), rtPS and nrtPS connections are not canceled within the simulate period.	32
Table 4.2	The Means for Packet Loss Rate vs Bandwidth.	58
Table 4.3	The Standard Deviations for Packet Loss Rate vs Bandwidth.	58
Table 4.4	The Confidence Intervals for Packet Loss Rate vs Bandwidth.	58
Table 4.5	The Means for Admission Blocking Probability vs Bandwidth.	60
Table 4.6	The Standard Deviations for Admission Blocking Probability vs Bandwidth.	60
Table 4.7	The Confidence Intervals for Admission Blocking Probability vs Bandwidth.	61
Table 4.8	The Means for Admission Blocking Probability vs Memory.	63
Table 4.9	The Standard Deviations for Admission Blocking Probability vs Memory.	63
Table 4.10	The Confidence Intervals for Admission Blocking Probability vs Memory.	63
Table 4.11	The Means for Useless Packet Rate vs Bandwidth.	64
Table 4.12	The Standard Deviations for Useless Packet Rate vs Bandwidth.	65
Table 4.13	The Confidence Intervals for Useless Packet Rate vs Bandwidth.	65

Table 4.14	The Means for Packet Loss Rate (no processing useless packets).	66
Table 4.15	The Standard Deviations for Packet Loss Rate (no processing useless packets).	66
Table 4.16	The Confidence Intervals for Packet Loss Rate (no processing useless packets).	67
Table 4.17	The Means for Packet Loss Rate vs Memory (Pushout scheme).	69
Table 4.18	The Standard Deviations for Packet Loss Rate vs Memory (Pushout scheme).	69
Table 4.19	The Confidence Intervals for Packet Loss Rate vs Memory (Pushout scheme).	69
Table 4.20	The Means for Packet Loss Rate vs Bandwidth (Pushout scheme).	70
Table 4.21	The Standard Deviations for Packet Loss Rate vs Bandwidth (Pushout scheme).	70
Table 4.22	The Confidence Intervals for Packet Loss Rate vs Bandwidth (Pushout scheme).	71

List of Figures

Figure 1.1	WiMax/802.16 architecture.	2
Figure 3.1	Buffer Management Architecture.	15
Figure 3.2	A graphic example for ConnectionsManager.	19
Figure 3.3	Status before a rtPS flow s is admitted into network.	20
Figure 3.4	Status after rtPS flow s is admitted into network.	20
Figure 3.5	Graphic example of memory allocation.	22
Figure 3.6	Status before discarding head packet.	25
Figure 3.7	Status after discarding head packet.	25
Figure 3.8	Status after an incoming packet enters into buffer.	26
Figure 4.1	A scheduled event list example.	32
Figure 4.2	Status before the connection m is canceled.	35
Figure 4.3	Status after the connection m is canceled.	36
Figure 4.4	Status before serving packets of connection j	38
Figure 4.5	Status after serving packets of connection j	38
Figure 4.6	Status when the burst remaining size is not less than the sum of the current packet size and minimum packet size, and it is not greater than the sum of the current packet size and maximum packet size.	49
Figure 4.7	Status when the burst remaining size is equal to the current packet size.	49
Figure 4.8	Status when the burst remaining size is greater than the sum of the current packet size and maximum packet size.	49
Figure 4.9	Packet Loss Rate vs Bandwidth.	57
Figure 4.10	Admission Blocking Probability vs Bandwidth.	60
Figure 4.11	Packet Loss Rate vs Memory.	61
Figure 4.12	Admission Blocking Probability vs Memory.	62
Figure 4.13	Useless Packet Rate vs Bandwidth.	64

Figure 4.14 Packet Loss Rate vs Bandwidth (no processing useless packets).	66
Figure 4.15 Packet Loss Rate vs Memory (Pushout scheme).	68
Figure 4.16 Packet Loss Rate vs Bandwidth (Pushout scheme).	70

Abstract

Quality of service (QoS) is a critical feature of WiMax/802.16. It guarantees a certain level of performance (bandwidth and delay) to a service flow in accordance with traffic contract parameters. WiMax/802.16 QoS guarantees are achieved by multiple management modules: base station (BS) scheduler, BS buffer manager, subscriber station (SS) scheduler and SS buffer manager. In the previous work, the first three management modules have been addressed, but SS buffer management for outgoing traffic is not mentioned. A WiMax/802.16 system without a SS buffer management scheme for outgoing traffic cannot completely provide QoS guarantees to multiple kinds of WiMax/802.16 service flows. The topic of this research is buffer management for outgoing traffic in a WiMax/802.16 SS. The first goal is to minimize the packet-loss-rate for real-time traffic flows. The second goal is to lower the overall packet-loss-rate caused by a lack of buffer space. The third goal is to achieve a QoS guarantee. In this thesis, we propose a new buffer management scheme called admission control and dynamic buffer allocation (ACDB) for outgoing traffic in a WiMax/802.16 SS. It consists of two parts: an admission control (AC) algorithm and a dynamic buffer allocation (DBA) algorithm. Simulation results demonstrate that our proposed buffer management scheme achieve the above three goals and provide QoS guarantee for multiple types of service flows.

Acknowledgements

I would like to thank Professor Michel Barbeau, my supervisor, for his guidance and many suggestions during this research. I am also thankful to Amy Tu, my wife, for her constant support.

Chapter 1

Introduction

1.1 Introduction

1.1.1 WiMax/802.16 Broadband Wireless Access Systems

WiMax/802.16 is a 4G wireless network technology [18]. It is a broadband wireless access (BWA) system designed for wireless metropolitan area networks (WMANs). It operates at the frequency bands of 2-11 GHz and 10-66 GHz with data rates of 32 to 130 Mbps depending on the channel width and modulation technique.

A WiMax/802.16 network consists of several subscriber stations (SSs) and a base station (BS). An example of WiMax/802.16 architecture is shown in Figure 1.1. There are two modes of operation for a WiMax/802.16 network: the point-to-multipoint (PMP) mode and mesh mode. In the PMP mode, a BS connects multiple SSs to a public network and controls all communications in the WiMax/802.16 network, e.g. a SS cannot directly communicate with any other SS. The association between a SS and the BS consists of two links: an uplink (from SS to BS) and a downlink (from BS to SS). Uplink transmission employs time division multiple access (TDMA), while downlink transmission depends on the time division multiplexing (TDM) scheme. On the downlink channel, the data packets from the BS are broadcasted to all SSs and each SS only picks up the packets destined to it. On the uplink, each SS transmits data packets in a specific time interval assigned by the BS indicated in an uplink bandwidth allocation map (UL-MAP).

WiMax/802.16 is connection oriented. It defines two kinds of connections: management and transport. Each connection is identified by a 16-bit connection ID (CID). The management connections consist of a basic, a primary and a secondary management connections. The basic connection is used for short and urgent management messages. The primary connection is for delay-tolerant management messages. The

Figure 1.1: WiMax/802.16 architecture.

secondary management connection is used for IP encapsulated management messages (e.g. DHCP, SNMP, TFTP). The basic and primary management connections are established after the completion of initial ranging, that determines the signal strength that must be used by a SS. The secondary management connection may be established after the registration process. Transport connections, which are used for user traffic flows, are dynamically established using the management message dynamic service addition (DSA). Each connection is associated with a service flow. Each service flow is unidirectional flow and characterized by a set of QoS parameters (such as bandwidth, latency and jitter requirements). It is identified by a 32-bit service flow ID (SFID) and has three states: provisioned, admitted and active. Only admitted and active service flows are provided with a CID. Service flows can be dynamically added, changed and deleted by using management messages such as dynamic service addition request (DSA-REQ), dynamic service change request (DSC-REQ) and dynamic service deletion (DSD-REQ).

WiMax/802.16 defines a physical (PHY) layer and a media access control (MAC) layer. At the PHY layer, the flow of bits is structured as a sequence of frames with the same length of 0.5, 1 or 2 ms. Each frame is divided into a number of minislots. There are two modes of operation: frequency division duplex (FDD) and time division duplex (TDD). For the FDD mode, every frame is divided into a downlink (DL)

subframe and an uplink (UL) subframe. The UL subframe and DL subframe are sent on different frequency bands simultaneously. An uplink subframe and downlink subframe have the same number of minislots. For the TDD mode, every frame is divided into an UL subframe and a DL subframe. The UL subframe follows the DL subframe, and they share the same wireless frequency band. The lengths of the UL and DL subframes are adjustable.

The MAC layer is comprised of three sublayers: service-specific convergence sublayer (CS), common part sublayer (CPS) and security sublayer. The CS interfaces the higher layer and CPS. It does connection classification. The CPS plays a critical role in the WiMax/802.16 protocol architecture. For instance, it controls the admission of new connections, manages the connections and service flows, provides QoS guarantee of connections and constructs protocol data units (PDUs). The WiMax/802.16 MAC layer defines QoS signaling mechanisms (e.g. connection set-up, bandwidth request, and the UL-MAP) and functions that control the data transmission between the BS and SSs (e.g. uplink scheduling for unsolicited grant service (UGS) flows). However, admission control, traffic policing, uplink scheduling for multiple types of service flows, and buffer management are not addressed in the WiMax/802.16 specification.

1.1.2 Traffic Model

In a WiMax/802.16 network, each uplink connection is mapped to a scheduling service flow. Wimax/802.16 supports four traffic scheduling services: UGS, real-time polling service (rtPS), non-real-time polling service (nrtPS), and best effort (BE). Different from the conventional telecommunications network where traffic follows exponential distribution [1], Wimax/802.16 traffic can not simply be modeled as an exponential distribution, but rather as self-similar processes [21].

UGS is designed to support real-time and constant bit rate (CBR) traffic flows, such as T1/E1 and voice over IP (VoIP) without silence suppression. It does not require any bandwidth request (BW-REQ) such as unicast polling, piggyback request or contention-based polling. In each time frame, a BS allocates a fixed size bandwidth to such a kind of traffic. For voice traffic, the interarrival time between two connections is exponentially distributed and a connection duration also follows the

exponential distribution [21].

The rtPS and nrtPS are designed to support variable bit rate (VBR) traffic flows. This type of traffic involves long bursts that are modeled as a Poisson Pareto burst process [25]. The arrival process of rtPS and nrtPS connections follows the exponential distribution. Burst interarrival time also follows the exponential distribution. Burst duration is modeled as a Pareto distribution [21].

The rtPS is designed to support real-time VBR traffic flows, such as MPEG video or VoIP with silence suppression. It has specific bandwidth and delay requirements. Whenever packets miss their maximum delay, they are useless. Because of their variable bit rate nature, rtPS flows have variable bandwidth demands. In each DL subframe, the BS provides SSs with unicast request opportunities via the UL-MAP. In the UL subframe, rtPS flows transmit their BW-REQs in the designated unicast request opportunities. Each BW-REQ includes the current queue size, which indicates the amount of bits that needs to be sent over the network in the next frame.

The nrtPS is designed to support non-real-time VBR traffic flows, such as FTP. It is time-insensitive and requires a minimum data rate. It uses periodic unicast request opportunities or contention request opportunities for transmitting its BW-REQ. A BW-REQ includes the current queue size.

The BE is designed to support data streams for which no minimum service level is required, such as HTTP and email. It does not have bandwidth or delay requirements. A connection request for this service is always admitted by the BS, regardless of the bandwidth availability. A BW-REQ for this service includes the current queue size and only depends on contention request opportunities for transmission. The type of traffic is self-similar [13]. The arrival process of BE connections is exponentially distributed. The document transfer size of BE traffic follows a Pareto distribution [21].

These scheduling services have different priorities. Ranking priorities from high to low, they are UGS, rtPS, nrtPS, and BE.

1.1.3 Quality of Service

QoS refers to control mechanisms that guarantee a certain level of performance to a data flow in accordance with requests from the application program. QoS guarantees

are characterized by bandwidth, delay, jitter (delay variation), and packet loss rate. For a data flow, its QoS is severely degraded if its packets incur insufficient bandwidth, high packet loss rate, large delays, and large delay variation.

QoS guarantees can be used to differentiate various kinds of traffic flows. In addition, QoS support in broadband wireless access (BWA) systems can potentially utilize bandwidth efficiently and provide additional revenues for wireless internet service providers (ISPs).

QoS is achieved with a buffer management unit and a scheduler controlling network resources (buffer and bandwidth) consumed by each flow according to a traffic contract (requested bandwidth and delay requirement).

WiMax/802.16 provides support for the QoS of data flows by defining QoS signaling mechanisms such as connection setup, bandwidth requests and bandwidth allocation. However, it does not define the uplink scheduling for all types of service flows, admission control, traffic policing process, and buffer management. WiMax/802.16 in itself does not completely and reliably guarantee QoS for data flows.

Wongthavarawat and Ganz [22] propose a new QoS architecture that extends functions defined in the IEEE 802.16 standard by adding admission control, traffic policing, and uplink scheduling for multiple types of service flows, but buffer management is not addressed. Chen et al. [6] mention the need for buffer management in WiMax/802.16, although they do not illustrate how buffer management works.

1.1.4 Buffer Management

A WiMax/802.16 SS's buffer for outgoing traffic is used to accommodate MAC service data units (SDU) that wait for bandwidth allocation for transmission over the network. Since a WiMax/802.16 network is connection oriented, a connection establishment is required before transmitting packets over the network. A SS can have several connections simultaneously. Packets from the higher layer are classified in the CS according to their CID. They are processed by a traffic policing module and put in an appropriate queue that shares the buffer memory with other queues. Because multiple queues share common buffer memory, there may be packet loss and degradation of QoS unless an appropriate buffer management scheme is used. For example,

in case packets from lower-priority service flows occupy the whole buffer space, the higher-priority packets maybe loss due to a lack of available space to accommodate them. It causes the QoS for the higher-priority service flows to degrade, even though there maybe sufficient bandwidth allocation for transmitting them. It is necessary to introduce buffer management schemes for the SS buffer for outgoing traffic in order to provide QoS guarantee in a WiMax/802.16 network.

Currently, there are three kinds of buffer management schemes: complete buffer partition (CBP), complete buffer sharing (CBS), and partial buffer sharing (PBS) [11]. There are two types of CBP schemes: static and dynamic. In the static scheme, the whole buffer memory is divided in multiple fixed size queues according to the differentiated classes of traffic, each of which corresponds to a single class. One queue cannot occupy the buffer space assigned to others. The assigned buffer space for each class cannot be modified as traffic load varies. The static scheme may cause low buffer utilization and high overall packet loss rate. The dynamic scheme extends the static scheme by adjusting buffer allocation according to the changing traffic load. The dynamic scheme significantly improves the buffer utilization and reduces the overall packet loss rate. The pushout scheme [9] is a typical CBS scheme. It lets a packet of any class freely enter into the shared buffer when there is available memory space, while it discards the lowest-priority packets when there is a lack of space and let the incoming higher-priority packets enter. The pushout scheme has the best performance when the traffic load is light, while the low-priority service flows have a high packet loss rate when the traffic load is heavy and memory capacity is limited. PBS is a buffer scheme that compromises the CBP and CBS schemes. Its performance is between the CBP and CBS schemes.

1.2 Motivation

The existing buffer management schemes CBP, CBS, and PBS [11] do not consider the maximum delay requirements of packets. For example, they do not deal with packets that become useless because they are not served within a maximum delay due to a lack of bandwidth. This results in waste of buffer space for new packet arrivals that are ready to enter into buffer, but there is no space available. Moreover,

they do not control admission of new connections. Packet are loss when no buffer space is available, but new connections are admitted. The goal of this research is to address these two weaknesses of CBP, CBS and PBS for outgoing traffic buffer management in a WiMax/802.16 SS.

1.3 Statement of the Problem

The WiMax/802.16 systems have strict QoS requirements for multiple types of service flows, especially for UGS and rtPS service flows. The existing buffer management schemes: CBP, CBS, and PBS cannot satisfy such strict QoS requirements because they have a high packet loss rate. These buffer management schemes do not process useless packets and control admission of new connections. Due to not collecting the buffer space occupied by useless packets, these buffer management schemes waste buffer space needed to accommodate new incoming packets. When the SS's memory capacity is small, this causes a high packet loss rate. In addition, due to the absence of an admission control mechanism, these buffer management schemes admit plenty of connections to the network, although there is no buffer space available, and thus cause a high packet loss. The goal of this thesis is to design a buffer management scheme for outgoing traffic in a WiMax/802.16 SS.

1.4 Contribution and Results

We propose a buffer management scheme that addresses the two weaknesses of CBP, CBS and PBS, minimizes real-time packet loss rates, decreases overall packet loss rate and achieves QoS guarantee for WiMax/802.16 service flows. Our proposed buffer management scheme is called the admission control and dynamic buffer allocation (ACDB) scheme. It contributes to decrease packet loss rate for multiple kinds of WiMax/802.16 service flows, to avoid real-time (UGS and rtPS) packet loss, and to provide QoS guarantees to these service flows. According to our simulation results, our proposed buffer management scheme significantly decreases the packet loss rate, especially high priority packet loss rate by using admission control and an useless

packet processing mechanisms. It even guarantees no UGS or rtPS packet loss, although the SS's memory capacity and assigned UL bandwidth may be small. As the assigned UL bandwidth and SS's memory capacity are increased, packet loss rates for the four types of service flows decrease. When the SS's memory capacity and assigned UL bandwidth are increased to some certain values, the packet loss rates for four types of service flows approach zero. Therefore, our proposed buffer management scheme can guarantee QoS for multiple types of WiMax/802.16 service flows and is suitable for being the buffer management scheme for outgoing traffic in a WiMax/802.16 SS.

1.5 Organization of Thesis

The remainder of this thesis is organized as follows. In Chapter 2, we review the related literature. In Section 2.1, we review the literature on buffer management. In Section 2.2, we review the literature on WiMax/802.16. In Chapter 3, we introduce our new buffer management scheme. In Section 3.1, we define the terminology used in this thesis. In Section 3.2, we describe the admission control algorithm. In Section 3.3, we describe the dynamic buffer allocation algorithm. In Chapter 4, we detail the simulation of our buffer management scheme. In Section 4.1, we describe the objectives of our simulation. In Section 4.2, we present the simulation environment. In Section 4.3, we discuss the simulation structure. In Section 4.4, we define the simulation algorithms. In Section 4.5, we show the simulation results. In Chapter 5, we conclude and discuss our future work.

Chapter 2

Literature Review

2.1 Buffer Management

Fan et al. [15] propose a hierarchical buffer management scheme for an ATM switch. This buffer management scheme divides the buffer memory into two different levels: ports and virtual connections (VCs). First, the whole buffer memory is allocated to multiple port queues. Second, each port queue is further divided into multiple VC queues. In each level, the buffer management scheme allows an overloaded queue to use any buffer memory not used by other queues through dynamically adjusting its queue threshold (the maximum amount of buffer a queue can access). This buffer management scheme ensures high buffer utilization and fair usage of buffer memory.

Choudhury and Hahne [10] propose a buffer management scheme called dynamic queue length threshold for shared-memory packet switches. This buffer management scheme lets the queue length threshold be proportional to the current amount of unused buffer in the switch at any instant of time. ATM cell arrivals for a queue are blocked whenever the queue length equals or exceeds the current threshold value. The buffer management scheme always holds a small amount of buffer space in reserve that is distributed equally among all queues. It adapts the queue threshold as traffic load varies. For instance, when a lightly loaded traffic flow suddenly becomes heavy, its corresponding queue and the total buffer occupancy increase, but the threshold is decreased. Any queue exceeding its threshold has its arrivals blocked until it frees some buffers and thus increases the queue threshold. This buffer management scheme improves the overall packet loss performance and fairness between traffic flows, but may lead to a waste of small amount of buffer space.

Chuang and Yin [11] propose a dynamic partial buffer sharing (DPBS) scheme. The DPBS scheme dynamically adjusts the discarding threshold based on the packet

loss behavior. In the DPBS scheme, arriving packets are classified according to multiple discarding priorities, and multiple discarding thresholds are deployed during the run time. An arriving packet is admitted into the queue only if its current occupancy is less than its discarding threshold; otherwise only high priority packets are admitted if the queue is not full, while low priority packets are discarded directly. Each discarding priority uses a packet loss counter to record the number of discarded packets. Each increment of the loss counter for high priority packets pushes the discarding threshold to decrease, while each increment of the loss counter for low priority packets pushes the discarding threshold to increase. The DPBS scheme achieves a low packet loss rate and a high buffer utilization.

Choudhury and Hahne [9] propose a selective pushout (SP) buffer management scheme for an ATM switch. This buffer management scheme allows an arriving cell to overwrite a cell with priority less than or equal to itself from the longest output queue in the buffer if the shared memory is full. It completely utilizes the buffer space in heavy load traffic condition. In addition, it offers a low overall cell loss rate and a very low loss rate for the high priority cells. However, it does not concern about fairness issue. High priority cells always have the privilege to use the shared buffer.

Yang et al. [23] propose a buffer management scheme called pushout with virtual thresholds (PVT) for ATM switches. The PVT scheme is the combination of the queue threshold scheme and SP scheme. The PVT scheme allows each queue to increase in length until its virtual threshold (VT). The following is the algorithm of the PVT scheme. Case 1: When the buffer is not full, all new arriving cells are allowed to freely enter the shared buffer without any restriction. Case 2: When the buffer is full, a high-priority arriving cell pushes out the low-priority cell at the head of the longest queue with length larger than its VT; if there is no low-priority cell, then the high-priority arriving cell pushes out the same-priority cell at the head of the longest queue; if the high-priority arriving cell is destined to the longest queue, it is discarded. When the queue length of a low-priority arriving cell is smaller than its VT, the arriving cell pushes out the same-priority cell at the head of the longest queue with a length larger than its VT; if there is no low-priority cell in this case, then the arriving cell pushes out the high-priority cell at the head of the longest queue. When

the queue length of a low-priority arriving cell is larger than its VT, it pushes out the same-priority cell at the head of the longest queue with length larger than its VT. If there is no same-priority cell in this case, then the arriving cell is discarded. This buffer management scheme dynamically achieves the fairness between traffic flows, low overall cell loss rate and very low loss rate of the high priority cells.

Rananand [20] proposes a hierarchical buffer management for ATM switches where the buffer is logically organized into multiple buffer levels: main, port, and class. In this hierarchical structure, the main and port levels use the same algorithm. The algorithm in the main level determines how much buffer space is allocated to each port. The buffer algorithm in the port level allocates an amount of buffer space for a port to each service class. Finally, the algorithm in the class level controls all VCs in the class to share the available buffer space fairly by dropping cells of those which occupy more buffer space than their fair share.

Alcaraz and Cerdan [2] describe a buffer management technique called slope based discard (SBD) for 3G wireless links. This buffer management scheme is an active queue management (AQM) techniques at the downlink buffer of radio link control (RLC). It discards packets depending on the observation of the filling rate of the buffer. When the buffer filling rate exceeds a certain value, a packet is dropped and buffer filling rate reduces.

Yao and Chen [24] propose a buffer management scheme for Fast handover in WiFi/802.11 networks. This buffer management mechanism is integrated into the original Fast Handover protocol. It allows to use buffers in both the previous access router (PAR) and the (new access router) NAR during a handoff process, and differentiates packets according to their traffic characteristics. These mechanisms improves the buffer utilization on routers and supports QoS services during a handoff process.

Citro et al. [12] propose a buffer management scheme called adaptive dynamic input buffer management (ADBM) for ATM networks. It makes the call processor dynamically alternate between fuzzy thresholding approach and fixed thresholding approach according to burstiness measure of the input traffic so as to achieve a higher throughput than the pure fixed thresholding scheme and a lower drop rate than the pure fuzzy thresholding scheme.

Chen and Li [7] propose a adaptive random early expiration detection (A-REED) based buffer management algorithm for real-time traffic over wireless networks. This scheme makes decision to discard a packet by predicting whether the packet can reach the receiver within its maximal delay bound. So it can prevent transmission of some useless packets from wasting wireless resources, decrease queue delay, and reduce expiration probability of following packets.

Ascia et al. [3] propose a fuzzy priority control (FPC) buffer management scheme for ATM and IP networks. By reserving some switch resources (i.e. buffer and link bandwidth) for high-priority flows, this scheme is able to prevent the low-priority traffic from degrading the QoS of high-priority so as to guarantee the QoS requirements of high-priority traffic flow traffic. In order to maximize the total throughput, the scheme allows unused reserved buffer resources to accommodate low-priority traffic flows.

A comparison between buffer management schemes in the related work is shown in the Table 2.1. DT [10] is a PBS-based buffer management scheme, which achieves low overall packet rate, fairness between traffic flows, and high buffer utilization. H-DCBP [20] is a CBP-based buffer management scheme, which achieves low high-priority packet loss rate, low overall packet loss rate, fairness between traffic flows, and high buffer utilization. DPBS [11] is a PBS-based buffer management scheme, which also achieves low high-priority packet loss rate, low overall packet loss rate, fairness between traffic flows, and high buffer utilization. H-DT [15] is a PBS-based buffer management scheme, which achieves fairness between traffic flows and high buffer utilization. PVT [23] is a PBS-based buffer management scheme, which achieves very the second lowest high-priority packet loss rate, low overall packet loss rate, fairness between traffic flows, and the second highest buffer utilization. SP [9] which achieves the lowest high-priority packet loss rate, low overall packet loss rate, fairness between traffic flows, and the highest buffer utilization . SBD [2] is an active queue management (AQM), which achieves high throughput and low queue delay. FHBM [24] is a buffer management scheme for WiFi/802.11 fast handover, which achieves high buffer utilization on routers and supports QoS during a handoff process. ADBM [12] is an adaptive and dynamic (AD) buffer management scheme, which achieves high

throughput and low overall packet loss rate. A-REED [7] is a improved random early detection (RED) buffer management scheme for real-time traffic flows, which concerns with high throughput and utilization. FPC [3] is PBS-based buffer management scheme, which achieves low high-priority packet loss rate, high utilization and high throughput.

Table 2.1: Comparisons of buffer management schemes. Note: high priority packet loss rate (HPLR), overall packet loss rate (OPLR), useless real-time packet handle (URTPH), access control (AC), dynamic queue threshold (DT), hierarchical dynamic complete buffer partition (H-DCBP), hierarchical dynamic queue threshold (H-DT), no consideration (N/C), fast handover buffer management (FHBm).

Scheme	Throughput	HPLR	OPLR	Fairness	Utilization	URTPH	AC
DT	N/C	N/C	low	yes	high	N/C	N/C
H-DCBP	N/C	low	low	yes	high	N/C	N/C
DPBS	N/C	low	low	yes	high	N/C	N/C
H-DT	N/C	N/C	N/C	yes	high	N/C	N/C
PVT	N/C	2 nd lowest	low	yes	2 nd highest	N/C	N/C
SP	N/C	lowest	low	no	highest	N/C	N/C
SBD	N/C	N/C	low	yes	high	N/C	N/C
FHBm	N/C	N/C	low	yes	high	N/C	N/C
ADBM	high	N/C	low	N/C	N/C	N/C	N/C
A-REED	high	N/C	N/C	no	high	N/C	N/C
FPC	high	low	N/C	no	high	N/C	N/C

2.2 WiMax/802.16

Hawa and Petr [16] propose a uplink scheduling architecture for the data over cable service interface specification (DOCSIS) and WiMax/802.16 that supports a QoS guarantee for various types of service flows. In addition, the authors also introduce a contention minislot allocation scheme that improves the scheduling performance under a variety of load conditions.

Wongthavarawat and Ganz [22] propose an uplink packet scheduling algorithm and an admission control policy for WiMax/802.16, and extend the IEEE 802.16 QoS architecture by adding an admission control module and a traffic policing module, and extending the uplink packet scheduling module.

Chen et al. [5] propose an integrated QoS architecture for the WiMax/802.16 TDD mode to support multi-layer QoS control and a mapping between the resource reservation protocol (RSVP) messages and WiMax/802.16 DSx messages (such as DSA, DSC and DSD) to implement a cross-layer QoS control.

Chen et al. [6] propose a WiMax/802.16 service flow management architecture for the TDD mode to dynamically adjust bandwidth allocation for uplinks and downlinks for achieving a higher throughput in an unbalanced traffic condition. Besides that, the authors also propose a deficit fair priority queue (DFPQ) scheduling scheme for multiple types of service flows.

Nair et al. [19] describe the WiMax/802.16 orthogonal frequency division multiplexing (OFDM), orthogonal frequency division multiple access (OFDMA), MAC protocols, types of provisioning, QoS, and the challenges inherent in implementing the MAC protocol.

Wang et al. [21] describe WiMax/802.16 traffic model, analyze upper bound blocking probability above the packet level for all types of traffic in WiMax/802.16 networks by using Gaussian model for aggregated traffic in large network and Chernoff bound method, and propose an admission control and bandwidth allocation mechanism above the packet level for WiMax/802.16 networks.

Eklund et al. [14] introduce the medium access control (MAC) and physical layer (PHY) features of the WiMax/802.16 standard, such as the UL-link and DL-link subframe structure, MAC PDU structure, service flow, and network entry.

Cho et al. [8] propose a new QoS architecture for the IEEE 802.16a. The proposed QoS architecture extends the original QoS architecture by adding a traffic management module at the SS, and adding a uplink bandwidth allocation scheduling (UBAS) module and admission control module at the BS. So it provides QoS support to real-time traffic flows. In addition, authors present a bandwidth allocation and admission control policy for the architecture.

Chapter 3

The Admission Control and Dynamic Buffer Allocation Scheme

In this thesis, we propose a new buffer management scheme called admission control and dynamic buffer allocation (ACDB) for outgoing traffic in a WiMax/802.16 SS. It consists of two parts: an admission control (AC) algorithm and a dynamic buffer allocation (DBA) algorithm. Their corresponding sub-modules construct a buffer management unit that, as a part of the extended WiMax/802.16 QoS architecture, is shown in the Figure 3.1.

Figure 3.1: Buffer Management Architecture.

The buffer management unit is used to control connection admission and memory allocation to packets. In addition, the buffer management unit also provides queue length of connections to the SS's BW-REQ generator at the end of each frame for requesting bandwidth.

3.1 Terminology

In this section, we define the terminology used in this thesis:

1. a_i is the number of bits of connection i that arrive during a time frame.
2. s_i is the number of bits of connection i that is served during a time frame.
3. r_i is the token bucket rate (average traffic rate) of connection i .
4. f is the duration of a frame (ms).
5. b_i is the token bucket size of connection i .
6. d_i is the maximum delay requirement of connection i .

Token bucket rate and token bucket size are two parameters for the token bucket algorithm, which is a rate control mechanism based on the presence of tokens in the bucket. Token bucket rate is the rate in which a token is added to the bucket. Token bucket size is the maximum number of tokens a bucket can hold. The maximum delay refers to the longest time a real-time packet keeps effective. If a real-time packet is not served before the maximum delay, it is useless.

3.2 Admission Control Algorithm

The admission control algorithm is used to restrict new connection admissions by checking whether there is sufficient buffer space available in the SS. It works together with the admission control module at the BS, which checks whether there is sufficient bandwidth in the BS. Since a WiMax/802.16 network is connection oriented, each application must establish a connection before transmitting its packets over the network. According to the work in Reference [22], the establishment of a new connection involves three stages: (1) An application that originates at a SS sends to the BS a connection request DSA-REQ including the traffic contract parameters (bandwidth and delay requirement). (2) The admission control module at the BS determines whether to accept or reject the new connection only based on whether the current available bandwidth can satisfy the bandwidth requirement of the new connection

and the QoS of the existing connections are not affected. (3) If the admission control module at the BS accepts the new connection, the BS assigns the connection a unique connection ID (CID) and provides the token bucket parameters to the traffic policing module at the SS.

However, alone this condition for the admission of a new connection is insufficient. For instance, a rapid increase of the packet loss rate and degradation of QoS of the service flows may occur when there is not sufficient buffer available for the arriving packets, but the system still admits new connections. In order to solve the problem, the admission control algorithm at each SS must simultaneously check whether there is sufficient available buffer space to accommodate the data packets for the new connection. If the two conditions are both satisfied, the new connection is admitted into the network. Otherwise, the process is done repeatedly until the new connection is established successfully. In this thesis, we focus the research on the second condition (check whether there is sufficient buffer space in a SS for admission of new connections).

For a UGS service flow, a_i in each frame is $f \times r_i$ bits. That means only $f \times r_i$ bits of buffer space is required to accommodate the service flow with rate of r_i . So $f \times r_i$ bits of buffer space is the minimum buffer requirement for admission of a new connection. If the current available buffer space (excluding the reserved buffer space) is not less than this size and there are sufficient available bandwidth in the BS for this service flow, a new connection associated to this service flow is allowed to be established.

For an rtPS service flow, under the control of the traffic policing module that is implemented by a token bucket algorithm, a_i in each time frame is not greater than $f \times r_i + b_i$ bits. So, the available buffer space in the SS must not be less than $f \times r_i + b_i$ bits if an rtPS service flow is ready to enter into the WiMax/802.16 network.

For a nrtPS service flow, because it generates variable size data packets, it is admitted to the WiMax/802.16 network as long as the available buffer space at a SS is not less than $f \times r_i + b_i$ bits.

For a BE service flow, it is always allowed to enter into the network. There are no memory requirements for its admission.

The subfunction $\text{admissionControl}(\text{flowType}, r_i, \text{remainingMemory})$ is shown in Algorithm 1. The three parameters of the function respectively refer to the type of a service flow that is requesting connection admission, average bandwidth required by the service flow, and current remaining memory for connection admission. The function initially calculates the corresponding memory requirement according to the type of the service flow, then compares the obtained value to the current remaining memory. If the current remaining memory is not less than the required memory for connection admission, the function returns true. Otherwise, the function returns false. If the function returns true, the request for connection admission is accepted by the system and a connection ID is allocated to the service flow. On the other hand, if the function returns false, the system rejects the requesting service flow to enter the network.

Algorithm 1 $\text{admissionControl}(\text{flowType}, r_i, \text{remainingMemory})$

```

if  $\text{flowType}$  is UGS then
     $\text{admissionRequiredMemory} \leftarrow r_i \times f$   $\{r_i$  and  $f$  respectively denote average
    bandwidth and frame duration $\}$ 
else if  $\text{flowType}$  is rtPS or nrtPS then
     $\text{admissionRequiredMemory} \leftarrow \text{bandwidth} \times f + b$   $\{b$  denotes token bucket size $\}$ 
else
     $\text{admissionRequiredMemory} \leftarrow 0$ 
end if
if  $\text{remainingMemory} \geq \text{admissionRequiredMemory}$  then
    return true  $\{\text{connection request is accepted}\}$ 
else
    return false  $\{\text{connection request is rejected}\}$ 
end if

```

Once a new connection is successfully established, it is added at the tail of an appropriate connection list that locates in a `connectionsManager`. The `connectionsManager` maintains four different connection lists: `UGSList`, `rtPSList`, `nrtPSList` and `BEList`. Each connection list manages one or multiple same-flow-type connections entered into the network. The `UGSList` is used for UGS connections. The `rtPSList` is used for rtPS connections. The `nrtPS` is used for nrtPS connections. `BEList` is used for BE connections. If no corresponding connection enter into the network, a connection list is empty. A graphic example of `ConnectionsManager` is shown in Figure 3.2.

This figure shows that seven connections have been established in a WiMax/802.16 network. Among them, connection j and l are of type UGS, connection n is of type rtPS, connection h , i , m are of type nrtPS, and connection k is of type BE.

Figure 3.2: A graphic example for ConnectionsManager.

Figure 3.3 gives an example to show the status (including the ConnectionsManager, remaining memory for new connection admission and a requesting service flow) before a rtPS service flow s is admitted to the network. This figure shows that remaining memory for new connection admission is greater than the admission requirement of service flow s . So it is admitted into the network.

Figure 3.4 shows the status (including the ConnectionsManager and remaining memory for new connection admission) after service flow s is admitted to enter into the network. In this figure, connection s is established and added at the tail of rtPSList. At the same time, the remaining memory for new connection admission decreases by the size of the admission requirement of service flow s .

3.3 Dynamic Buffer Allocation Algorithm

The dynamic buffer allocation algorithm is mainly responsible for controlling the buffer space size for each connection, and handling packet dropping when packets are useless, or the buffer is full. In this scheme, buffer space is dynamically allocated to the established connections according to the type and traffic contract parameters.

Figure 3.3: Status before a rtPS flow s is admitted into network.

Figure 3.4: Status after rtPS flow s is admitted into network.

The required buffer space for a connection is determined by the packet arrival process and packet service process. The packet arrival process is influenced by the traffic contract rate, while the packet service process is affected by the bandwidth allocated by the scheduler.

For a UGS service flow, it generates $f \times r_i$ bits in each frame, and the BS scheduler grants $f \times r_i$ bits of bandwidth allocation to it in each frame. That means a_i and s_i for a UGS service flow are the same. Once the connection is established successfully, the DBA scheme reserves $f \times r_i$ bits of buffer space for this connection, which cannot be shared with other connections unless the service flow associated with this connection is deleted.

For the remaining three kinds of service flows, because the a_i and bandwidth allocated by the scheduler in each time frame are variable and hard to determine, the buffer allocation scheme allows them to share the remaining available buffer space. Whenever the buffer is not full, the new arrivals will enter the buffer without any restriction. If the buffer is full, the DBA scheme tries to find the lowest-priority and longest-length queue and discards the packets at the head of that queue. If updated available memory is not enough to accommodate the incoming packet, it continues to find the longest-length queue and discards its head packets. If updated available memory is still less than the incoming packet size after discarding all packets from the lowest priority, the DBA scheme does the same for lower priority connections until the available memory is enough to accommodate the incoming packet. If the updated available memory after discarding packets from all connections whose priorities are not greater than the one of the incoming packet, the new incoming packet is discarded. For instance, when an rtPS incoming packet is ready to enter into the buffer, but the buffer is full. Firstly, the DBA scheme finds the lowest priority in the connectionsManager. If the lowest-priority is BE, then the DBA scheme finds the longest-queue connection. After discarding the head packets of the longest-queue connection, the updated available memory is not enough for the incoming rtPS packet, the DBA scheme continues to find the next longest-queue connection and discard its head packets. After all packets from BE connections are discarded, the updated available memory is still less than the incoming rtPS packet. The DBA scheme continues to find the lowest priority in the connectionsManager. If the the lowest priority is nrtPS, then the DBA scheme finds the longest-queue connection among all nrtPS connections and discards its head packets until the updated available memory is enough

for the incoming packet. If the updated available memory is not less than the incoming packet size, the incoming rtPS packet is allowed to enter into the buffer. After discarding all packets from connections whose priorities are not greater than rtPS, if the updated available memory is still not enough to accommodate the incoming rtPS packet, it is discarded.

Figure 3.5 provides a graphic example about how memory is allocated to different types of connections. In this figure, three UGS connections: *UGS 1*, *UGS 2* and *UGS 3* are respectively allocated a fixed amount of memory that is not shared with other connections. The connections: *rtPS 1*, *rtPS 2*, *nrtPS 1*, *nrtPS 2*, *BE 1* and *BE 2* share the rest of memory. The unused shared memory is called available memory. If a new rtPS, nrtPS, or BE service flow is admitted into the network, its packets also share this part of memory as long as available memory is sufficient. If a new UGS connection is admitted, it reserves an amount of memory from the shared memory. The reserved size depends on the connection's bandwidth.

Figure 3.5: Graphic example of memory allocation.

The functions related to the DBA are reviewed in the following.

The subfunction `findLowPriority(lowerPriority)` (shown in Algorithm 2) is used to find the lowest priority among the connections whose priority are higher than the priority specified by the parameter. If the parameter is equal to 4, then this subfunction returns 5. That means the priority specified by the parameter is the

highest priority (UGS) and the lowest priority does not exist. If the parameter is less than 4, it searches the connection lists whose priority is equal to the parameter plus one. For example, if the parameter is 2 (nrtPS), then the subfunction firstly searches the rtPSList in the connections manager. If the rtPSList is not empty, then its corresponding priority is the lowest priority. Otherwise, the subfunction searches the UGSList. If in the process of searching, no such lowest priority is found, then the subfunction returns 5.

Algorithm 2 findLowPriority(lowerPriority)

```

lowPriority  $\leftarrow$  0
if lowerPriority == 4 then
    lowPriority = lowerPriority + 1
else
    while lowerPriority <= 4 do
        lowPriority = lowPriority + 1
        list = getList(lowPriority)
        if !list.isEmpty() then
            return lowPriority
        else
            lowerPriority  $\leftarrow$  lowPriority
        end if
    end while
end if
return lowPriority

```

The subfunction findLongestConnection(connectionList) (shown in Algorithm 3) is used to find the longest-queue connection in a connection list. It firstly assigns the first connection in the connectionList as the longest-queue connection. Then it compares the queue length of the longest-queue connection to the one of the current connection. If the queue length of the longest-queue connection is less, then it assigns the current connection as the longest-queue connection. Then it iteratively compares the longest-queue connection to the next connection in the connectionList until the list reaches the last connection.

Whenever a packet is ready to enter into buffer, but the available memory is not enough for it, the subfunction discardHeadPackets(availableMemory, connection, packetSize) (shown in Algorithm 4) is invoked. This subfunction is used to discard the

Algorithm 3 findLongestConnection(*connectionList*)

```

longestConnection  $\leftarrow$  connectionList.connection
while connectionList.connection is not empty do
    currentConnection  $\leftarrow$  connectionList.connection
    queueLength  $\leftarrow$  currentConnection.queueLength
    if longestConnection.queueLength < queueLength then
        longestConnection  $\leftarrow$  connectionList.connection
    end if
    connectionList.next
end while
return longestConnection

```

head packets from the specified connection queue to provide enough memory for the incoming high-priority packet. Its three parameters respectively refer to the current available memory for incoming packets, longest-queue connection, and size of the incoming packet. The subfunction iteratively executes the operation of removing the packets at the head of the connection queue until the available memory is enough to accommodate an incoming packet. Once a packet is removed from the corresponding connection queue, the queue length decreases by the length of the removed packet, and available memory size increases by the length of the removed packet.

Algorithm 4 discardHeadPackets(*availableMemory*, *connection*, *packetSize*)

```

while availableMemory < packetSize do
    connection.queue.remove(headPacket)
    connection.queue.queueLength  $- =$  headPacket.size
    availableMemory  $+=$  headPacket.size
    connection.queue.next
end while

```

Figure 3.6 gives an example that shows the status (including the Connections-Manager, available memory and an incoming packet) before the head packet of a connection is discarded when an incoming packet from a higher-priority connection is ready to enter into the buffer, but the available memory is not enough to accommodate the incoming packet. In this figure, the available memory is not enough to accommodate the incoming packet from the UGS connection *l*, the lowest-priority is BE, and connection *k* has the longest queue length. So the head packet of connection *k* is discarded.

Figure 3.6: Status before discarding head packet.

Figure 3.7 shows the status (including the ConnectionsManager, available memory) after the head packet of connection k is discarded. In this figure, the head packet of connection k is discarded from its queue, queue length of connection k decreases by the size of the head packet, and available memory increases by the corresponding size.

Figure 3.7: Status after discarding head packet.

Whenever a packet is accepted into a buffer, the subfunction `addAndUpdateMemory(availableMemory, connection, packetSize)` (shown in Algorithm 5) is invoked. The execution of the subfunction causes an incoming packet to be put into an appropriate connection queue, increases the corresponding queue length by the size of the incoming packet, and decreases the available memory by the size of the incoming packet.

Algorithm 5 `addAndUpdateMemory(availableMemory, connection, packetSize)`

```

packet  $\leftarrow$  new Packet(packetSize)
connection.queue.add(packet) {let packet enter appropriate queue}
connection.queueLength  $+=$  packetSize {update queue length}
availableMemory  $- =$  packetSize {update available memory size}

```

Figure 3.8 give an example that shows the status (including ConnectionsManager, available memory) after an incoming packet enters into the buffer. In this figure, the incoming packet is put into the queue of connection l , queue length of connection l increases by the incoming packet size, and available memory decreases by the corresponding size.

Figure 3.8: Status after an incoming packet enters into buffer.

The subfunction `getList(priority)` (shown in Algorithm 6) is used to get a list in the ConnectionsManager corresponding to a priority (flow type) specified by the parameter.

Algorithm 6 getList(priority)

```

if priority is BE then
  return ConnectionsManager.BEList
else if priority is nrtPS then
  return ConnectionsManager.nrtPSList
else if priority is rtPS then
  return ConnectionsManager.rtPSList
else
  return ConnectionsManager.UGSList
end if

```

When a non-UGS packet is ready to enter into the buffer, the subfunction `bufferAllocation(availableMemory, connection, packetSize)` (shown in Algorithm 7) is invoked. If the specified available memory is enough to accommodate the incoming packet whose size is specified by a parameter, it calls the subfunction `addAndUpdateMemory(availableMemory, connection, packetSize)`. Otherwise, it firstly calls the subfunction `findLowPriority()` to get the lowest priority from the existing priorities. If the obtained lowest priority is greater than the priority associated to the incoming packet, the incoming packet is discarded. Otherwise, the subfunction `getList(lowestPriority)` and `getLongestConnection(list)` are invoked to get a longest-queue connection. Then the subfunction `discardHeadPackets(availableMemory, longestConnection, packetSize)` is invoked to discard the head packet(s) from the longest connection queue. If the updated memory *avMem* is not less than the incoming packet size, then the subfunction `addAndUpdateMemory(avMem, connection, packetSize)` is invoked to let the incoming packet enter into an appropriate connection queue. Otherwise, it continues to find the longest-queue connection. If there is such a longest-queue connection, then it repeats the above procedure. Otherwise, it tries to find the second lowest priority and repeat the above procedure.

When a packet is ready to enter into the buffer, the subfunction `dynamicBufferAllocation(availableMemory, connection, packetSize)` (shown in Algorithm 8) is invoked. It first check which flow type an incoming packet belongs to. If the packet is of type UGS, then the subfunction checks whether the reserved memory for its corresponding

Algorithm 7 *bufferAllocation(availableMemory, connection, packetSize)*

```

if availableMemory  $\geq$  packetSize then
    addAndUpdateMemory(availableMemory, connection, packetSize)
else
    lowerPriority  $\leftarrow$  0
    while lowerPriority < 4 do
        lowPriority  $\leftarrow$  findLowPriority(lowerPriority)
        if connection.priority  $\geq$  lowPriority then
            list  $\leftarrow$  getList(lowPriority)
            while true do
                longestCon  $\leftarrow$  getLongestConnection(list)
                if longestCon  $\neq$  null then
                    avMem  $\leftarrow$ 
                        discardHeadPackets(availableMemory, longestCon, packetSize)
                    if avMem  $\geq$  packetSize then
                        addAndUpdateMemory(avMem, connection, packetSize)
                        return
                    end if
                else
                    break
                end if
            end while
            lowerPriority  $\leftarrow$  lowPriority
        else
            discardthepacket {discard incoming packet}
            break
        end if
    end while
end if
return

```

connection is enough for the packet. If yes, the subfunction *addAndUpdateMemory(availableMemory, connection, packetSize)* is invoked to accommodate the incoming packet. Otherwise, the packet is discarded. If the packet is of non-UGS, the subfunction *bufferAllocation(availableMemory, connection, packetSize)* is invoked.

For each connection, it has a maximum delay requirement. Different types of connection corresponds to different maximum delay requirements. If a packet from some connection is not transmitted when its maximum delay is reached, it becomes an useless packet. Storage and transmission of useless packets waste precious network

Algorithm 8 *dynamicBufferAllocation(availableMemory, connection, packetSize)*

```

if connection.flowType is UGS then
    memory  $\leftarrow$  connection.reservedMemory
    queueLength  $\leftarrow$  connection.queueLength
    if memory  $\geq$  packetSize then
        addAndUpdateMemory(memory, connection, packetSize)
        connection.setReservedMemory(memory - packetSize) {update reserved
        memory}
    else
        discardthepacket
    end if
else
    bufferAllocation(availableMemory, connection, packetSize)
end if

```

resources (buffer space and bandwidth). In order to utilize the buffer space occupied by useless packets and increase the available buffer space for new incoming packets, the useless packets are removed from the buffer in our proposed buffer management scheme. The number of removed useless packets is not counted in the total packet loss because the rate of useless packets is influenced by the total bandwidth of the BS and scheduling algorithm, not by the memory capacity and buffer management scheme.

The subfunction *handleUselessPacket(list)* (shown in Algorithm 9) is used to remove all useless packets from all connections in a connection list. If the deadline time of a packet is less than the sum of the current event time and *f* (a frame duration), the packet is considered useless. After removing the useless packets, this subfunction invokes the subfunction *releaseMemory(packetSize)* to release the occupied memory.

Algorithm 9 *handleUselessPacket(list)*

```

numberOfConnection  $\leftarrow$  list.size()
for i = 0 to numberOfConnection do
  connection  $\leftarrow$  list.get(i)
  queue  $\leftarrow$  connection.queue
  flowType  $\leftarrow$  connection.flowType
  while !queue.isEmpty() do
    packet  $\leftarrow$  queue.getFirst()
    deadlineTime  $\leftarrow$  packet.deadlineTime
    packetSize  $\leftarrow$  packet.packetSize
    if deadlineTime < eventTime + f then
      queue.remove(packet)
      if flowType == UGS then
        connection.reservedBuffer + = packetSize
      else
        releaseMemory(packetSize)
      end if
    else
      break
    end if
  end while
end for

```

Chapter 4

Simulation of the Admission Control and Dynamic Buffer Allocation Scheme

4.1 Simulation Objective

The objective of our simulation is to analyze the number of lost bits vs the size of buffer capacity, to analyze the number of lost bits vs the UL bandwidth assigned to a SS, and to compare our algorithm with other buffer management schemes.

4.2 Simulation Environment

We use Java 2 to implement the simulation. The simulation period is 3600 seconds (simulate time). In the simulation, we assume that the frame duration is 10 ms and token bucket size is 1000 bytes (the maximum packet size). The other parameters used in the simulation are given in Table 4.1. The values of bandwidth and delay in the table are chosen in accordance with Reference [6]. The bandwidth mentioned in our simulation refers to the UL bandwidth assigned to a SS in each frame duration. The generation of connection requests for each type of service flow follows an exponential distribution with a mean of 10 seconds. For a UGS connection, its duration is exponentially distributed with a mean of 60 seconds. For a BE connection, its duration follows a Pareto distribution with the shape parameter 1.5 and scale parameter $200 \times 8/r_i$, where the value of 200 bytes denotes the minimum size of a packet and r_i denotes the bandwidth of connection i . For a rtPS or nrtPS connection, its burst interarrival time follows an exponential distribution with a mean of 10 seconds, while its burst duration follows a Pareto distribution with the shape parameter 1.5 and scale parameter $200 \times 8/r_j$, where r_j denotes the bandwidth of connection j . The generation of packets is either self-similar or exponentially distributed. During the entire simulation period, the packets in the buffer are scheduled once per second.

Table 4.1: Simulation Parameters. Note: average connection interarrival (ACI), average connection duration (ACD), average burst interarrival (ABI), average burst duration (ABD), rtPS and nrtPS connections are not canceled within the simulate period.

Type	Bandwidth (kbps)	Delay (ms)	ACI (s)	ACD (s)	ABI (s)	ABD (s)
UGS	1-3	10	10	60		
rtPS	6-10	20-70	10	3600	10	1
nrtPS	3-6	60-150	10	3600	10	1
BE	1-3	160-280	10	10		

4.3 Simulation Structure

Our simulation is a discrete-event simulation that is based on the event scheduling scheme [4]. During the simulate period, once an event is activated, its next occurrence is scheduled at some event time and put into the schedule event list according to the time order. The event time for each event is not real time, but simulation milliseconds. The smallest-scheduled-time event is always the first in the schedule event list. An example of schedule event list is shown in the Figure 4.1. In this figure, the first event is an AdmissionControl event from a BE service flow whose event time is 8.5. After this event is activated, it first schedules a PacketArrival event to immediately occur, then schedules the next AdmissionControl event to occur at the simulate time of 18.3. Because the time of the next AdmissionControl event is the largest, it is added at the tail of the scheduled event list.

Figure 4.1: A scheduled event list example.

Whenever the execution of an event in the schedule event list finishes, 1) the event is removed from the scheduled event list, 2) the simulation time is updated to the time of the next event in the scheduled event list, 3) the next event is executed. For instance, in Figure 4.1, when the simulation time is 8.5, the first AdmissionControl

event is executed. After completion, it is removed from the scheduled event list. Because a BE PacketArrival event is located at the head of the scheduled event list, the simulation time is updated to this event time. And because the new event time is also 8.5, the simulation time is still 8.5. Then the PacketArrival event is executed. The events that follow in the scheduled event list are handled the same way. The subfunction `runSimulation(period)` (shown in Algorithm 10) describes this process. The period parameter in Algorithm 10 specifies the duration of a simulation.

Algorithm 10 `runSimulation(period)`

```

while !scheduledEventList.isEmpty() do
    event  $\leftarrow$  scheduledEventList.getFirst()
    simulatedTime  $\leftarrow$  event.eventTime
    if simulatedTime > period then
        break
    end if
    event.actions()
    scheduledEventList.remove(ev)
end while
stopSimulation()

```

The simulation involves five events: `WimaxEvent`, `AdmissionControl`, `PacketArrival`, `ConnectionCancel`, and `PacketSchedule`. Among them, `WimaxEvent` is a super class, the other four events are derived from it. An outline of the `WimaxEvent` is shown in Page 34.

The `actions()` method represents the actions associated with an event. These actions are executed when the event occurs. An event is scheduled to occur at a specified simulation time by calling its `schedule()` method. The desired event time is passed as an argument to the method [17].

Whenever a service flow is ready to establish a connection, an `AdmissionControl` event occurs. The occurrence of the `AdmissionControl` event for some type of flow is exponentially distributed. The class `AdmissionControl` includes three instance fields: traffic bandwidth, traffic maximum delay and traffic flow type. Each instance of this class may have different values for bandwidth, maximum delay and flow type. In addition, the class implements the `actions()` method that is illustrated in Algorithm 24 in Subsection 4.4.1. An outline of the class `AdmissionControl` is shown in Page

```

public abstract class WimaxEvent {
    private static double simulatedTime = 0.0;
    protected double eventTime = 0.0;
    private static LinkedList < WimaxEvent > scheduledEventList
        = new LinkedList < WimaxEvent > ();

    protected abstract void actions();

    public final void schedule(double eventTime);

    public final static void runSimulation(double period);

    public final static void stopSimulation();
}

```

34.

```

public class AdmissionControl extends WimaxEvent {
    private int bandwidth;
    private int maximumDelay;
    private ServiceFlowTypes flowType;

    public AdmissionControl(type, bw, delay) {
        bandwidth = bw;
        maximumDelay = delay;
        flowType = type;
    }

    public void actions();
}

```

Whenever a connection is canceled, a ConnectionCancel event occurs. The cancellation time of a connection is determined by two factors: the time of its AdmissionControl event t and its connection duration d . As soon as a connection is successfully established, a ConnectionCancel event is scheduled to occur at simulation time $t + d$. The class ConnectionCancel includes only one instance field: connection. The field specifies which connection is canceled. The actions() method is also implemented in this class. An outline of the class ConnectionCancel is shown in Page 35.

```

public class ConnectionCancel extends WimaxEvent {
    private Connection connection;

    public ConnectionCancel(connection) {
        this.connection = connection;
    }

    public void actions();
}

```

Figure 4.2 gives an example to show the status (including the ConnectionsManager, available memory and remaining memory for new connection admission) before a connection is canceled. In this figure, connection m whose queue includes two packets is going to be discarded.

Figure 4.2: Status before the connection m is canceled.

Figure 4.3 shows the status (including ConnectionsManager, available memory and remaining memory for new connection admission) after the connection m is canceled. In this figure, the connection m is removed from BEList in the ConnectionsManager, its connection queue is cleared and its occupied memory is released, available memory increases by the size of the released memory, and remaining memory for new connection admission increases by the size of the admission requirement of connection m .

Figure 4.3: Status after the connection m is canceled.

Whenever a packet from some connection arrives, a **PacketArrival** event occurs. For a UGS or a BE connection, the first **PacketArrival** event immediately occurs after the occurrence of an **AdmissionControl** event. For a rtPS or nrtPS connection, the first **PacketArrival** event occurs at the beginning of the first burst. The production of a burst follows an exponential distribution. The occurrence of other **PacketArrival** events is explained in its **actions()** method. The class **PacketArrival** includes three instance fields: **connection**, **burst** and **packet size**. There are two constructors in this class. The selection of the constructor depends on the flow type of a connection. If a packet is from a rtPS or nrtPS connection, the second constructor is used to construct a **PacketArrival** instance. Otherwise, the first constructor is used. This class also implements the **actions()** method, which shows in Algorithm 30 in Subsection 4.4.1. An outline of the class **PacketArrival** is shown in Page 37.

At the end of each frame duration, a **PacketSchedule** event occurs. Once a **PacketSchedule** event occurs, packets in each connection queue are served according to priority. The class **PacketSchedule** includes only one instance field: **available bandwidth**. The field refers to SS's UL bandwidth assigned by the BS. The **actions()**

```

public class PacketArrival extends WimaxEvent {
    private Connection connection;
    private Burst burst;
    private int packetSize;

    public PacketArrival(connection, packetSize) {
        this.connection = connection;
        this.burst = null;
        this.packetSize = packetSize;
    }

    public PacketArrival(connection, burst, packetSize) {
        this.connection = connection;
        this.burst = burst;
        this.packetSize = packetSize;
    }

    public void actions();
}

```

method is also implemented in this class. An outline of the class PacketSchedule is shown in Page 37.

```

public class PacketSchedule extends WimaxEvent {
    private int availableBandwidth;

    public PacketSchedule(bandwidth) {
        availableBandwidth = bandwidth;
    }

    public void actions();
}

```

Figure 4.4 gives an example to show the status (including the ConnectionsManager, available memory and available bandwidth) before packets of connection j are served. In this figure, the available bandwidth is greater than the queue length of connection j , so all packets of connection j can be served.

Figure 4.5 shows the status (including ConnectionsManager, available memory and available bandwidth) after packets of connection j are served. In this figure,

Figure 4.4: Status before serving packets of connection j .

the queue of connection j is cleared, queue length becomes zero, available memory increases by the queue length of connection j , and available bandwidth decreases by the corresponding size.

Figure 4.5: Status after serving packets of connection j .

An outline of the `main()` function of our simulation is shown in Page 39. Firstly, it

iteratively produces the first AdmissionControl Event for the four types of service flow. For each type of service flow, different bandwidth and maximum delay are randomly produced as the parameters of the AdmissionControl Event. The first connection request occurrence time for each type of service flow starts from the simulation time of 0.0, and follows an exponential distribution with a mean of 10000 ms (see in Table 4.1). After that, the first PacketSchedule event is scheduled at the simulation time 10 ms (a frame duration). Its parameter is *bandwidthofSS* that refers to the total bandwidth assigned to a SS. Finally, the subfunction *runSimulation(SIMULATION_PERIOD)* is invoked to process each event in the scheduled event list according to time order.

```

public static void main(String[] args) {
    startTime = 0.0;
    for(i = 1; i <= 4; i++) {
        if(i == 1)
            flowType = UGS;
        else if(i == 2)
            flowType = rtPS;
        else if(i == 3)
            flowType = rtPS;
        else
            flowType = BE;
        bandwidth = nextBandwidth(flowType);
        maxDelay = maxDelay(flowType);
        AdmissionControl initialEvent =
            new AdmissionControl(flowType, bandwidth, maxDelay);
        startTime += nextExponential(10000);
        initialEvent.schedule(startTime);
        startTime = 0.0;
    }
    new PacketSchedule(bandwidthofSS).schedule(startTime + f);
    runSimulation(SIMULATION_PERIOD);
}

```

4.4 Simulation Algorithms

4.4.1 Algorithms for Our Proposed Buffer Management Scheme

This subsection reviews all algorithms of our proposed buffer management scheme used in the simulation.

The `schedule(evTime)` method (shown in Algorithm 11) defined in the class `WimaxEvent` is used to schedule an event (`AdmissionControl`, `PacketArrival`, `ConnectionCancel` or `PacketSchedule`) to occur at some simulation time. Its parameter specifies the event time of an event. This method is inherited by all subclasses and is not required to be overwritten.

Algorithm 11 `schedule(evTime)`

```

if evTime < simulatedTime then
    throw new RuntimeException("attempt to schedule past event");
end if
eventTime  $\leftarrow$  evTime
listSize  $\leftarrow$  scheduledEventList.size()
if (listSize == 0) then
    scheduledEventList.add(0, this)
else
    j  $\leftarrow$  listSize - 1
    while (j >= 0) do
        WimaxEvent ev  $\leftarrow$  scheduledEventList.get(j)
        if (ev.eventTime > eventTime) && (j != 0) then
            j --
        else if (ev.eventTime > eventTime) && (j == 0) then
            scheduledEventList.add(j, this)
            break
        else
            scheduledEventList.add(j + 1, this)
            break
        end if
    end while
end if

```

The `stopSimulation` method (shown in Algorithm 12) of Class `WimaxEvent` is invoked whenever no event in the schedule event list or the simulated period is reached. It is used to clear all events in the schedule event list and reset the simulate time as

zero.

Algorithm 12 stopSimulation()

```

while !scheduledEventList.isEmpty() do
    event  $\leftarrow$  scheduledEventList.getFirst()
    scheduledEventList.remove(event)
    simulatedTime  $\leftarrow$  0
end while

```

The function nextPareto(scale, shape) (shown in Algorithm 13) is used to randomly produce a Pareto distributed value. The scale parameter specifies the minimum value of the produced variable. The shape parameter is set to 1.5, according to ns-2. In our simulation, this function is mainly invoked to produce the BE connection duration and burst duration for a rtPS/nrtPS connection.

Algorithm 13 nextPareto(scale, shape)

```

if scale  $\leq$  0 then
    new throw RuntimeException("scale is not a positive number")
end if
randx  $\leftarrow$  nextDouble()
power  $\leftarrow$  Math.pow(1 - randx, 1/shape)
return scale/power

```

The function exponential(time) (shown in Algorithm 14) is used to randomly produce an exponential distributed random value. The time parameter denotes the mean of the variable. In our simulation, this function is mainly invoked to produce connection interarrival times, UGS connection durations and burst interarrival times for a rtPS/nrtPS connection.

Algorithm 14 exponential(time)

```

randx  $\leftarrow$  nextDouble()
result  $\leftarrow$  -1 * time * Math.log(randx)
return result

```

The function nextBandwidth(flowType) (shown in Algorithm 15) is used to randomly produce a bandwidth for a service flow. The flowType parameter specifies the type of a service flow. According to [6], the UGS bandwidth is 1-3 bits per millisecond, rtPS bandwidth is 6-10 bits per millisecond, nrtPS bandwidth is 3-6 bits per

millisecond, and BE bandwidth is 1-3 bits per millisecond.

Algorithm 15 nextBandwidth(flowType)

```

if flowType == UGS then
    bandwidth = randInt(1, 3)
else if flowType == rtPS then
    bandwidth = randInt(6, 10)
else if flowType == nrtPS then
    bandwidth = randInt(3, 6)
else
    bandwidth = randInt(1, 3)
end if
return bandwidth

```

The function maxDelay(flowType) (shown in Algorithm 16) is used to randomly produce a maximum delay for a service flow. The flowType parameter specifies the type of a service flow. According to [6], the maximum delay for UGS is one frame duration, the maximum delay for rtPS is 2-7 frame duration, the maximum delay for nrtPS is 6-15 frame duration, the maximum delay for BE is 16-24 frame duration.

Algorithm 16 maxDelay(flowType)

```

if flowType == UGS then
    maximumDelay = f {f denote a frame duration}
else if flowType == rtPS then
    maximumDelay = randInt(2, 7) * f
else if flowType == nrtPS then
    maximumDelay = randInt(6, 15) * f
else
    maximumDelay = randInt(16, 24) * f
end if
return maximumDelay

```

Whenever a UGS service flow is admitted into a network, the subfunction reservedMemoryForUGS(flowType, reservedSize) (shown in Algorithm 17) is invoked to assign a fixed-length memory to its associated connection. The assigned memory length is specified by the reservedSize parameter.

The subfunction reserveMemoryForConnection(requiredSize) (shown in Algorithm 18) is used to reserve an amount of memory for a connection. The requiredSize parameter specifies the size of the reserved memory. The variable totalReservedMemory

Algorithm 17 reservedMemoryForUGS(flowType, reservedSize)

```

if flowType == UGS then
    reservedBufferSize  $\leftarrow$  reservedSize
    availableMemory  $- =$  reservedSize
else
    reservedBufferSize  $\leftarrow$  0
end if

```

refers to total memory reserved for all existing connections.

Algorithm 18 reserveMemoryForConnection(requiredSize)

```

totalReservedMemory+ = requiredSize

```

Whenever a connection is successfully established, the subfunction addConnectionAndReserveMemory(connection) (shown in Algorithm 19) is invoked to add the connection into one of the lists in the ConnectionsManager. In addition, in order to guarantee the QoS of the connection, it also reserves an amount of memory for the connection according to its memory requirement by invoking the subfunction reserveMemoryForConnection(requiredSize). In the function body, the symbols f and b respectively denote frame duration and token bucket size.

Algorithm 19 addConnectionAndReserveMemory(connection)

```

flowType  $\leftarrow$  connection.flowType
if flowType == UGS then
    ConnectionsManager.UGSList.add(connection)
    connectionRequiredMemory  $\leftarrow$  bandwidth *  $f$ 
    reservedMemoryForUGS(flowType, connectionRequiredMemory)
else if flowType == rtPS then
    ConnectionsManager.rtPSList.add(connection)
    connectionRequiredMemory  $\leftarrow$  bandwidth *  $f + b$ 
else if flowType == nrtPS then
    ConnectionsManager.nrtPSList.add(connection)
    connectionRequiredMemory  $\leftarrow$  bandwidth *  $f + b$ 
else
    ConnectionsManager.BEList.add(connection)
    connectionRequiredMemory  $\leftarrow$  0
end if
reserveMemoryForConnection(connectionRequiredMemory)

```

The subfunction scheduleNextAdmissionControl() (shown in Algorithm 20) is used

to schedule the next AdmissionControl event. The scheduled event time is the sum of the current event time and the connection interarrival time. The connection interarrival time follows an exponential distribution with mean meanACInterarrival, which is 10 seconds. The variable of eventTime denotes the current event time.

Algorithm 20 scheduleNextAdmissionControl()

```

nextACTime  $\leftarrow$  exponential(meanACInterarrival) + eventTime
nextBW  $\leftarrow$  random.nextBandwidth(flowType)
nextDelay  $\leftarrow$  random.maxDelay(flowType)
nextAC  $\leftarrow$  new AdmissionControl(flowType, nextBW, nextDelay)
nextAC.schedule(nextACTime)

```

The function nextPoisson(double lambda) (shown in Algorithm 21) is used to randomly produce a Poisson distributed value. The parameter denotes the mean of the variable. In our simulation, this function is invoked to produce a burst rate.

Algorithm 21 nextPoisson(double lambda)

```

elambda  $\leftarrow$  Math.exp(-1 * lambda)
product  $\leftarrow$  1
count  $\leftarrow$  0
while product  $\geq$  elambda do
    product  $\ast$  nextDouble()
    result  $\leftarrow$  count
    count ++
end while
return result

```

The subfunction scheduleFirstPacketArrival() (shown in Algorithm 22) is used to schedule the first PacketArrival event. For a rtPS/nrtPS connection, the occurrence of the first PacketArrival event immediately follows the occurrence of the first burst. The first packet size is determined by the first burst length. The burst length is the product of the burst duration and burst rate. The burst duration follows a Pareto distribution with the shape parameter 1.5 and scale parameter $(200 \times 8)/bandwidth$, where 200 denotes the minimum packet size. If the burst length is greater than the size of the maximum packet, then we randomly produce a packet size that guarantees that, after the transmission of that packet, the remaining of the burst is not less than the size of a minimum packet. If the burst length is between the size of the minimum

packet and the size of the maximum packet, then the packet size is the burst length. For a BE connection, the first packet size is any integer between 200 and 1000 bytes. For a UGS connection, the size for all packets is 500 bits.

Algorithm 22 `scheduleFirstPacketArrival()`

```

flowType  $\leftarrow$  connection.flowType
if (flowType == rtPS) || (flowType == nrtPS) then
    burstInterarrival  $\leftarrow$  exponential(meanBurstInterarrival)
    scale  $\leftarrow$  (200 * 8) / bandwidth
    burstDuration  $\leftarrow$  nextPareto(scale, SHAPE)
    burstRate  $\leftarrow$  nextPoisson(bandwidth)
    burstLength  $\leftarrow$  burstDuration * bandwidth
    burst  $\leftarrow$  new Burst(burstLength, burstRate)
    longestLen  $\leftarrow$  1000 * 8
    shortestLen  $\leftarrow$  200 * 8
    if burstLength > longestLen then
        while true do
            packetSize  $\leftarrow$  randInt(200, 1000) * 8
            if (burstLength - packetSize) >= shortestLen then
                break
            end if
        end while
    else
        packetSize  $\leftarrow$  burst.burstDuration
    end if
    packetArrival  $\leftarrow$  new PacketArrival(connection, burst, packetSize)
    packetArrival.schedule(burstInterarrival + eventTime)
else if flowType == BE then
    packetSize  $\leftarrow$  randInt(200, 1000) * 8
    newPacketArrival(connection, packetSize).schedule(eventTime)
else
    packetSize = UGS_PACKET_SIZE
    newPacketArrival(connection, packetSize).schedule(eventTime)
end if

```

The subfunction `scheduleConnectionCancel()` (shown in Algorithm 23) is used to schedule a `ConnectionCancel` event. The event time is the sum of the connection admission time and connection duration. The variable of `eventTime` in the function body denotes the connection admission time. For a UGS connection, its connection duration is an exponential variable with the mean `meanUGSDuration`, which is shown

in Table 4.1. For a BE connection, the connection duration is a Pareto variable. For a rtPS/nrtPS connection, the connection cancelation time is the simulation period.

Algorithm 23 *scheduleConnectionCancel()*

```

flowType  $\leftarrow$  connection.flowType
if flowType == UGS then
    connectionCancelTime  $\leftarrow$  exponential(meanUGSDuration) + eventTime
else if flowType == BE then
    scale  $\leftarrow$  meanBEDuration * (SHAPE - 1) / SHAPE
    connectionCancelTime  $\leftarrow$  nextPareto(scale, SHAPE) + eventTime
else
    connectionCancelTime  $\leftarrow$  SIMULATION_PERIOD
end if
nextConnectionCancel  $\leftarrow$  new ConnectionCancel(connection)
nextConnectionCancel.schedule(connectionCancelTime)

```

The *actions()* method (shown in Algorithm 24) for the *AdmissionControl* class is executed when an *AdmissionControl* event occurs. It firstly invokes the subfunction *admissionControl(flowType, bandwidth, remainingMem)*. If the return value is true, 1) an associated connection is established; 2) the subfunction *addConnectionAndReserveMemory(connection)* is invoked to add the connection into one of the connection lists in the *ConnectionsManager* and reserve the connection required memory; 3) the subfunction *scheduleFirstPacketArrival()* is invoked to schedule the first *PacketArrival* event; 4) the subfunction *scheduleConnectionCancel()* is invoked to schedule the *ConnectionCancel* event. No matter whether the current connection is admitted or not to the network, the subfunction *scheduleNextAdmissionControl()* is invoked to schedule the next *AdmissionControl* event.

Algorithm 24 *actions() for AdmissionControl*

```

isAdmitted  $\leftarrow$  admissionControl(flowType, bandwidth, remainingMem)
if isAdmitted then
    connection  $\leftarrow$  new Connection(flowType, bandwidth, maximumDelay)
    addConnectionAndReserveMemory(connection)
    scheduleFirstPacketArrival()
    scheduleConnectionCancel()
end if
scheduleNextAdmissionControl()

```

The subfunction `produceNextPacketSize()` (shown in Algorithm 25) is used to randomly produce a size for the next packet, which must guarantee that the burst remaining size is not less than the minimum packet size.

Algorithm 25 `produceNextPacketSize()`

```

while true do
  nextPacketSize  $\leftarrow$  randInt(200, 1000) * 8
  if (nextBurstLen - nextPacketSize)  $\geq$  shortestLen then
    break
  end if
end while

```

The subfunction `scheduleNextRTorNRTPacket()` (shown in Algorithm 26) is used to schedule the next packet for a rtPS/nrtPS connection. (1) If the burst remaining size is not less than the sum of the current packet size and minimum packet size, and it is not greater than the sum of the current packet size and maximum packet size (shown in Figure 4.6), then the next packet size is set to the burst remaining minus the current packet size. The next PacketArrival event time is self-similar and equal to the division of the current packet size by the bandwidth. (2) If the burst remaining size is equal to the current packet size (shown in Figure 4.7), the next packet is the first packet of the next burst. In this case, the next PacketArrival event time is equal to the occurrence time of the next burst. The next burst follows an exponential distribution with a mean of 10 seconds. The next packet size depends on the produced next burst length. The next burst length is the product of the next burst duration and corresponding burst rate. The next burst duration follows a Pareto distribution with the shape parameter 1.5 and scale parameter $(200 \times 8)/\text{bandwidth}$, where 200 denotes the minimum packet size. (3) If the burst remaining size is greater than the sum of the current packet size and maximum packet size, (shown in Figure 4.8), then the subfunction `produceNextPacketSize()` is invoked to produce the next packet size, which guarantees that the burst remaining size is not less than the size of a minimum packet. The next PacketArrival event time is self-similar and equal to the division of current packet size by the bandwidth.

Figure 4.6 gives an example to show status when the burst remaining size is greater than the sum of `packetSize` and `shortestLen`, and less than the sum of `packetSize` and

Algorithm 26 `scheduleNextRTorNRTPacket()`

```

longestLen  $\leftarrow 1000 * 8$ 
shortestLen  $\leftarrow 200 * 8$ 
remaining  $\leftarrow$  burst.remaining
if (packetSize + shortestLen  $\leq$  remaining  $\leq$  packetSize + longestLen) then
    packetInterarrivalTime  $\leftarrow$  packetSize / bandwidth
    burst.remaining  $- =$  packetSize
    nextPacketSize  $\leftarrow$  burst.remaining
    nextPAEvent  $\leftarrow$  new PacketArrival(connection, burst, nextPacketSize)
    nextPAEvent.schedule(eventTime + packetInterarrivalTime)
else if burst.remaining  $==$  packetSize then
    burstInterarrival  $\leftarrow$  exponential(meanBurstInterarrival)
    scale  $\leftarrow 200 * 8 / \textit{bandwidth}$ 
    nextBurstDuration  $\leftarrow$  nextPareto(scale, SHAPE)
    nextBurstRate  $\leftarrow$  nextPoisson(bandwidth)
    nextBurstLen  $\leftarrow$  nextBurstDuration * nextBurstRate
    nextBurst  $\leftarrow$  new Burst(nextBurstLen, nextBurstRate)
    if nextBurstLen > longestLen then
        produceNextPacketSize()
    else
        nextPacketSize = nextBurst.remaining
    end if
    nextPAEvent  $\leftarrow$  new PacketArrival(connection, nextBurst, nextPacketSize)
    nextPAEvent.schedule(burstInterarrival + eventTime)
else if burst.remaining > packetSize + longestLen then
    burst.remaining  $- =$  packetSize
    produceNextPacketSize()
    nextPAEvent  $\leftarrow$  new PacketArrival(connection, burst, nextPacketSize)
    nextPAEvent.schedule(eventTime + packetInterarrivalTime)
end if

```

longestLen.

Figure 4.7 gives an example to show status when the burst remaining size is equal to *packetSize*.

Figure 4.8 gives an example to show the status when the burst remaining size is greater than the sum of *packetSize* and *longestLen*.

The subfunction `scheduleNextPacketArrivalEvent()` (shown in Algorithm 27) is used to schedule the next *PacketArrival* event. If the packet belongs to a UGS flow, the packet interarrival time is self-similar and equal to the division of the current

Figure 4.6: Status when the burst remaining size is not less than the sum of the current packet size and minimum packet size, and it is not greater than the sum of the current packet size and maximum packet size.

Figure 4.7: Status when the burst remaining size is equal to the current packet size.

Figure 4.8: Status when the burst remaining size is greater than the sum of the current packet size and maximum packet size.

packet size by the bandwidth. The next PacketArrival event time is equal to the packet interarrival time. Because a UGS flow has a constant packet size, the next

packet size is equal to the current packet size. If the packet belongs to a rtPS/nrtPS flow, the subfunction `scheduleNextRTorNRTPacket()` is invoked. If the packet belongs to a BE flow, the packet interarrival time is the same as for a UGS flow. The next packet size is between 200 and 1000 bytes.

Algorithm 27 `scheduleNextPacketArrivalEvent()`

```

flowType  $\leftarrow$  connection.flowType
if flowType == UGS then
    packetInterarrivalTime  $\leftarrow$  packetSize/bandwidth
    nextPAEvent  $\leftarrow$  new PacketArrival(connection, packetSize)
    nextPAEvent.schedule(eventTime + packetInterarrivalTime)
else if (flowType == rtPS) || (flowType == nrtPS) then
    scheduleNextRTorNRTPacket()
else
    packetInterarrivalTime  $\leftarrow$  packetSize/bandwidth
    nextPacketSize  $\leftarrow$  randInt(200, 1000) * 8
    nextPAEvent  $\leftarrow$  new PacketArrival(connection, nextPacketSize)
    nextPAEvent.schedule(eventTime + packetInterarrivalTime)
end if

```

The subfunction `getPriority(connection)` (shown in Algorithm 28) is used to get the corresponding service flow type through the specified connection.

Algorithm 28 `getPriority(connection)`

```

flowType  $\leftarrow$  connection.flowType
if flowType == UGS then
    return UGS
else if flowType == rtPS then
    return rtPS
else if flowType == nrtPS then
    return nrtPS
else
    return BE
end if

```

The subfunction `isIncluded(connection)` (shown in Algorithm 29) is used to check whether a connection is maintained in `ConnectionsManager`. It finds the appropriate connection list in the `ConnectionsManager` through the flow type of the connection. Then it checks whether the connection is in the connection list. If yes, it returns true. Otherwise, it returns false.

Algorithm 29 *isIncluded(connection)*

```

priority  $\leftarrow$  getPriority(connection)
list  $\leftarrow$  getList(priority)
sizeOfList  $\leftarrow$  list.size()
for i = 0 to sizeOfList do
    con  $\leftarrow$  list.get(i)
    if con.connectionID == connection.connectionID() then
        return true
    end if
end for
return false

```

The *actions()* method (shown in Algorithm 30) for *PacketArrival* class is executed when a *PacketArrival* event occurs. If the connection of the incoming packet is in the *ConnectionsManager* (the connection is not canceled), the method calls the subfunction *dynamicBufferAllocation(availableMemory, connection, packetSize)* to allocate memory for the incoming packet or discard it. Then it calls the subfunction *scheduleNextPacketArrivalEvent()* to schedule the next *PacketArrival* event.

Algorithm 30 *actions() for PacketArrival*

```

if isIncluded(connection) then
    dynamicBufferAllocation(availableMemory, connection, packetSize)
    scheduleNextPacketArrivalEvent()
end if

```

The subfunction *releaseMemoryForConnection(connectionRequiredMem)* (shown in Algorithm 31) is used to release memory reserved for a connection. The released memory size is specified by the parameter. After releasing, the total memory reserved by all existing connections decreases by the released memory size, while the remaining memory for new connection admission increases by the released memory size.

Algorithm 31 *releaseMemoryForConnection(connectionRequiredMem)*

```

totalMemoryForConnections  $- =$  connectionRequiredMem
remainingMemoryForAdmission  $+ =$  connectionRequiredMem

```

The subfunction *releaseMemory(memorySize)* (shown in Algorithm 32) is used to release memory. The size of the released memory is specified by the parameter. After releasing, the available memory increases by the released memory size.

Algorithm 32 *releaseMemory(memorySize)*
availableMemoryForPackets += *memorySize*

The actions() method (shown in Algorithm 33) for ConnectionCancel is executed when a ConnectionCancel event occurs. It first calculates the required memory for the connection according to the flow type and connection bandwidth. If the connection is of type UGS, the required memory for the connection is released. If the connection is of other type, the connection queue is cleared and its occupied memory is released. For any type of connection, it is removed from its corresponding connection list and its reserved memory is released after it is canceled.

Algorithm 33 *actions() for ConnectionCancel*

```

flowType  $\leftarrow$  connection.flowType
if flowType == UGS then
    connectionRequiredMemory  $\leftarrow$  bandwidth * f {f is a frame duration}
    releaseMemory(connectionRequiredMemory)
else if flowType == BE then
    connectionRequiredMemory  $\leftarrow$  0
    releaseMemory(connection.queueLength)
else
    connectionRequiredMemory  $\leftarrow$  bandwidth * f + b {b is token bucket size}
    releaseMemory(connection.queueLength)
end if
priority  $\leftarrow$  getPriority(connection)
LinkedList < Connection > list  $\leftarrow$  getList(priority)
list.remove(connection)
releaseMemoryForConnection(connectionRequiredMemory)

```

The subfunction packetService() (shown in Algorithm 34) is used to serve all packets in the buffer according to the priority order. UGS packets are served first, then rtPS packets, nrtPS packets, and BE packets.

The subfunction packetServiceInList(list) (shown in Algorithm 35) is used to serve packets of connections in the connection list. The connection list is specified by the parameter. If the list is UGSList, the reserved memory for each UGS connection and available bandwidth are updated after the packets are served. If the list is other type of list, the available memory and bandwidth are updated after the packets are served. The subfunction returns a boolean value. If it returns true, that means that

Algorithm 34 packetService()

```

priority  $\leftarrow$  UGS
while priority  $\geq$  BE do
    LinkedList  $\langle$  Connection  $\rangle$  list  $\leftarrow$  getList(priority)
    isThereBandwidth  $\leftarrow$  packetServiceInList(list)
    if isThereBandwidth then
        break
    end if
    priority --
end while

```

there are still bandwidth available to serve other packets. Otherwise, no bandwidth is available.

Algorithm 35 packetServiceInList(list)

```

listSize  $\leftarrow$  list.size()
i  $\leftarrow$  0
while i < listSize do
    connection  $\leftarrow$  list.get(i)
    queue  $\leftarrow$  connection.queue
    queueLength  $\leftarrow$  queue.queueLength
    while !queue.isEmpty() do
        packet  $\leftarrow$  queue.getFirst()
        packetSize  $\leftarrow$  packet.packetSize
        if packetSize  $\leq$  availableBandwidth then
            queue.remove(packet)
            if list == UGSList then
                reservedMemory + = packetSize
            else
                releaseMemory(packetSize)
            end if
            queueLength -- = packetSize
            availableBandwidth -- = packetSize
        else
            return false {no bandwidth available}
        end if
    end while
    i ++
end while
return true {there is still bandwidth available}

```

The `actions()` method (shown in Algorithm 36) of the `PacketSchedule` class is executed when an instance of `PacketSchedule` occurs. It firstly invokes the subfunction `packetService()` to serve the packets in the buffer. Then it iteratively invokes the subfunction `handleUselessPacket(list)` to remove all useless packets in the buffer. Finally, it schedules the next `PacketSchedule` event by calling the `schedule()` method.

Algorithm 36 *actions() for PacketSchedule*

```

packetService() {packet scheduling function}
priority  $\leftarrow$  4
while priority  $\geq$  1 do
    LinkedList  $\leftarrow$  Connection  $>$  list  $\leftarrow$  getList(priority)
    handleUselessPacket(list)
    priority  $-$  1
end while
nextPSEvent  $\leftarrow$  new PacketSchedule(bandwidth)
nextPSEvent.schedule(eventTime + f) {f is a frame duration}

```

4.4.2 Algorithms for the Pushout Buffer Management Scheme

This subsection reviews all algorithms of the Pushout buffer management scheme used in the simulation. The pushout buffer management scheme does not use admission control nor an useless packet handling mechanisms.

The subfunction `findLowestPriority()` (shown in Algorithm 37) is used to find the lowest priority among all connections admitted into the network. It first checks whether `BEList` in the `connectionsmanager` contains connections. If yes, `BE` is the lowest priority. Otherwise, it subsequently checks `nrtPSList`, `rtPSList` and `UGSList`. Once it finds a connection list is not empty, it returns the priority associated with the connection list as the lowest priority.

The function `pushoutBufferManagement(availableMemory, connection, packetSize)` (shown in Algorithm 38) is associated with Pushout buffer management scheme. If the current available buffer space is enough to accommodate an incoming packet with size `packetSize`, it invokes the subfunction `addAndUpdateMemory(availableMemory, connection, packetSize)` (shown in Algorithm 5) to let the incoming packet enter into

Algorithm 37 findLowestPriority()

```

if !ConnectionsManager.BEList.isEmpty() then
    return BE
else if !ConnectionsManager.nrtPSList.isEmpty() then
    return nrtPS
else if !ConnectionsManager.rtPSList.isEmpty() then
    return rtPS
else
    return UGS
end if

```

the buffer. Otherwise, it finds the lowest priority among all connections by calling subfunction findLowestPriority(). Then it gets the connection list associated with the lowest priority by calling the subfunction getList(lowestPriority) (shown in Algorithm 6). And then it finds the longest-queue connection in the lowest-priority connection list by calling the subfunction getLongestConnection(list) (shown in Algorithm 3). And then it invokes the subfunction discardHeadPackets(availableMemory, longestCon, packetSize) (shown in Algorithm 4) to discard the head packets of the longest-queue connection. After discarding, if the updated available buffer space is not less than the size of the incoming packet, the subfunction addAndUpdateMemory(availableMemory, connection, packetSize) is invoked to let the incoming packet enter into the buffer. Otherwise, the incoming packet is discarded.

Algorithm 38 pushoutBufferManagement(availableMemory, connection, packetSize)

```

if availableMemory >= packetSize then
    addAndUpdateMemory(availableMemory, connection, packetSize)
else
    lowestPriority ← findLowestPriority()
    list ← getList(lowestPriority)
    longestCon ← getLongestConnection(list)
    avMem ← discardHeadPackets(availableMemory, longestCon, packetSize)
    if avMem >= packetSize then
        addAndUpdateMemory(avMem, connection, packetSize)
    else
        discardtheincomingpacket
    end if
end if

```

4.5 Simulation Results

In this section, we present diagrams illustrating the relations between packet loss rate and bandwidth, packet loss rate and memory, admission blocking probability and bandwidth, admission blocking probability and memory, and useless packet rate and bandwidth. In addition, we compare our proposed buffer management scheme with a scheme not handling useless packets. We compare our proposed buffer management scheme with the pushout buffer management scheme, which has the lowest packet loss rate and highest buffer space utilization among the existing buffer management schemes.

Figure 4.9 shows four curves (Packet Loss Rate vs. Bandwidth) when the memory capacity is set to 1 M bits. The four curves correspond to UGS, rtPS, nrtPS and BE service flows. UGS and rtPS curves show that the packet loss rates of UGS and rtPS service flows are always zero. As the bandwidth is increased, packet loss rates are null. BE and nrtPS curves show that, when the bandwidth assigned to a SS is less than 100 K bits, packet loss rates are very high. The packet loss rates for BE and nrtPS service flows respectively exceed 90% and 70%. The packet loss rate of BE is higher than the one of nrtPS. This is because bandwidth is always used to serve high priority packets first. When the bandwidth assigned to a SS is insufficient, plenty of lower priority packets in the SS's buffer are not served due to a lack of bandwidth and thus the entire buffer is occupied. At that time, if new incoming packets enter into the buffer, either low priority packets in the buffer or low priority incoming packets are discarded. If the incoming packets are high priority, the low priority packets in the buffer are discarded. If the incoming packets are low priority and no lower priority packets are in the buffer, the incoming packets are discarded. These two cases cause low priority packet loss. When the assigned bandwidth exceeds 100 K bits, low priority (such as nrtPS and BE) packet loss rates begin to decrease significantly. Once the assigned bandwidth is greater than 500 K bits, their packet loss rates drop to zero.

Each Y-value in Figure 4.9 is the mean of a sample of size 16 when the SS's memory capacity is set to 1 M bits and the assigned UL bandwidth is a specific value, the X-axis. $LR_1, LR_2, LR_3, \dots, LR_{16}$ are the values of a random sample. Each value is

Figure 4.9: Packet Loss Rate vs Bandwidth.

determined by a normally distributed random variable X with unknown expectation and unknown standard deviation. LR_i denotes the estimation of packet loss rate at time i .

Using equations

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i \quad (1)$$

and

$$S^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2 \quad (2)$$

We can get the means (shown in Table 4.2) and standard deviations (shown in Table 4.3) of packet loss rate for the four types of service flows when the SS's memory capacity is set to 1 M bits.

T in the following equation

$$T = \frac{\bar{X} - \mu}{S/\sqrt{n}} \quad (3)$$

Table 4.2: The Means for Packet Loss Rate vs Bandwidth.

Bandwidth	UGS LR(%)	rtPS LR(%)	nrtPS LR(%)	BE LR(%)
50K (bits)	0	0	84.8	93.3
100K (bits)	0	0	70.5	91.2
200K (bits)	0	0	1.3	15.7
500K (bits)	0	0	0	0
1M (bits)	0	0	0	0
2M (bits)	0	0	0	0

Table 4.3: The Standard Deviations for Packet Loss Rate vs Bandwidth.

Bandwidth	UGS LR(%)	rtPS LR(%)	nrtPS LR(%)	BE LR(%)
50K (bits)	0	0	2.5	2.1
100K (bits)	0	0	7.3	2.7
200K (bits)	0	0	0.8	7
500K (bits)	0	0	0	0
1M (bits)	0	0	0	0
2M (bits)	0	0	0	0

follows a Student-t distribution. For a 90% confidence level, then

$$Pr[-c < T < c] = 90\% \quad (4)$$

Substituting Equation (3) into Equation (4), the Equation (4) changes to

$$Pr[\bar{X} - c \frac{S}{\sqrt{n}} < \mu < \bar{X} + c \frac{S}{\sqrt{n}}] = 90\% \quad (5)$$

Using Equation (5), we get the confidence intervals (shown in Table 4.4) of the packet loss rate for the four types of service flows when the SS's memory capacity is set to 1 M bits.

Table 4.4: The Confidence Intervals for Packet Loss Rate vs Bandwidth.

Bandwidth	UGS LR(%)	rtPS LR(%)	nrtPS LR(%)	BE LR(%)
50K (bits)	0	0	84.8 ± 1.1	93.3 ± 0.9
100K (bits)	0	0	70.5 ± 3.2	91.2 ± 1.2
200K (bits)	0	0	1.3 ± 0.4	15.7 ± 3
500K (bits)	0	0	0	0
1M (bits)	0	0	0	0
2M (bits)	0	0	0	0

The results in Figure 4.9 indicate that the packet loss rate for a SS is influenced by the assigned UL bandwidth. The influence is especially significant for low priority service flows (such as BE and nrtPS). The larger the assigned bandwidth, the lower is the packet loss rate. The lower the assigned UL bandwidth, the larger is the packet loss rate. In order to decrease the packet loss rate, it is necessary to increase the UL

bandwidth assigned to a SS. In addition, the results in Figure 4.9 indicate that our proposed buffer management scheme can, although the assigned bandwidth may be not sufficient, keep real-time packet loss to an extremely low level once connections are admitted into the network. Therefore our proposed buffer management scheme can guarantee QoS for multiple types of service flows when the UL bandwidth assigned to a SS is adequate.

Figure 4.10 shows a curve (Admission Blocking Probability vs. Bandwidth) when the memory capacity is set to 1 M bits. As the assigned bandwidth is increased, the curve fluctuates between 51% and 55.5% blocking probability. This result indicates that the admission blocking probability is not related to the assigned bandwidth where the buffer size is the only factor for connection admission. When the remaining buffer space for connection admission is not less than the connection admission requirement of a service flow, a new connection is admitted to the network. Otherwise the connection request is blocked. In a real system, however, admission blocking probability is significantly affected by the assigned bandwidth. For instance, if the assigned UL bandwidth is insufficient at some specific time, a connection request is not admitted although the available buffer space is sufficient. Therefore, besides considering the remaining buffer size for connection admission, bandwidth also requires attention in real system.

Each Y-value in Figure 4.10 is the mean of a sample of size 16 when the SS's memory capacity is set to 1 M bits and the assigned UL bandwidth is a specific value, the X-axis. $ABP_1, ABP_2, ABP_3, \dots, ABP_{16}$ are the values of a random sample. Each value is determined by a normally distributed random variable X with unknown expectation and unknown standard deviation. ABP_i denotes the estimation of admission blocking probability at time i .

Using the Equations (1), (2) and (5), we can get the means, standard deviations and 90% confidence intervals (shown in Tables 4.5, 4.6 and 4.7) of admission blocking probability when the SS's memory capacity is set to 1 M bits.

Figure 4.11 shows four curves (Packet Loss Rate vs. Memory) when the assigned UL bandwidth is set to 1 M bits. The four curves respectively correspond to UGS, rtPS, nrtPS and BE service flows. For the four types of service flows, their packet

Figure 4.10: Admission Blocking Probability vs Bandwidth.

Table 4.5: The Means for Admission Blocking Probability vs Bandwidth.

Bandwidth	Admission Blocking Probability(%)
50K (bits)	55
100K (bits)	55.1
200K (bits)	54.8
500K (bits)	52.6
1M (bits)	51.4
2M (bits)	54.8

Table 4.6: The Standard Deviations for Admission Blocking Probability vs Bandwidth.

Bandwidth	Admission Blocking Probability(%)
50K (bits)	4.4
100K (bits)	4.1
200K (bits)	4.9
500K (bits)	4.3
1M (bits)	3.8
2M (bits)	3.7

loss rates are always zero regardless of the SS's memory capacity.

Each Y-value in Figure 4.11 is the mean of a sample of size 16 when the assigned UL bandwidth is set to 1 M bits and the SS's memory capacity is a specific value, the X-axis. $LR_1, LR_2, LR_3, \dots, LR_{16}$ are the values of a random sample. Each value is determined by a normally distributed random variable X with unknown expectation

Table 4.7: The Confidence Intervals for Admission Blocking Probability vs Bandwidth.

Bandwidth	Admission Blocking Probability(%)
50K (bits)	55 ± 1.9
100K (bits)	55.1 ± 1.8
200K (bits)	54.8 ± 2.1
500K (bits)	52.6 ± 1.9
1M (bits)	51.4 ± 1.7
2M (bits)	54.8 ± 1.6

Figure 4.11: Packet Loss Rate vs Memory.

and unknown standard deviation. LR_i denotes the estimation of packet loss rate at time i . Because the means are always zero, the standard deviations and confidence intervals are also zero.

The results in Figure 4.11 indicate that our proposed buffer management scheme provides an extremely low packet loss rate for multiple types of service flows as long as the bandwidth assigned to a SS in each frame duration is sufficient, although the memory capacity for outgoing traffic in a SS is not large. If a SS provides an appropriate memory capacity for outgoing traffic and the BS assigns a sufficient UL bandwidth in each frame duration, then our buffer management scheme even provides a zero packet loss rate.

Figure 4.12 shows a curve (Admission Blocking Probability vs. Memory) when the assigned UL bandwidth is set as 1 M bits. According to the curve in this figure, the admission blocking probability decreases as the SS's memory capacity is increased.

Figure 4.12: Admission Blocking Probability vs Memory.

Each Y-value in Figure 4.12 is the mean of a sample of size 16 when the assigned UL bandwidth is set to 1 M bits and the SS's memory capacity is a specific value, the X-axis. $ABP_1, ABP_2, ABP_3, \dots, ABP_{16}$ are the values of a random sample. Each value is determined by a normally distributed random variable X with unknown expectation and unknown standard deviation. The ABP_i denotes the estimation of admission blocking probability at time i .

Using Equations (1), (2) and (5), we get the means, standard deviations and 90% confidence intervals (shown in Tables 4.8, 4.9 and 4.10) of admission blocking probability when the assigned UL bandwidth is set to 1 M bits.

The results in Figure 4.12 indicate that the admission blocking probability for a SS is impacted by the SS's memory capacity for outgoing traffic. The larger the memory capacity, the lower is the admission blocking probability. If a SS could provide an infinite size buffer for outgoing traffic, its admission blocking probability

Table 4.8: The Means for Admission Blocking Probability vs Memory.

Capacity	Admission Blocking Probability(%)
100K (bits)	62.3
200K (bits)	58.4
300K (bits)	58.3
400K (bits)	58.2
500K (bits)	58
1M (bits)	53.6
2M (bits)	46.5

Table 4.9: The Standard Deviations for Admission Blocking Probability vs Memory.

Capacity	Admission Blocking Probability(%)
100K (bits)	4.4
200K (bits)	2.8
300K (bits)	4.8
400K (bits)	4.5
500K (bits)	3.4
1M (bits)	3.8
2M (bits)	3.2

Table 4.10: The Confidence Intervals for Admission Blocking Probability vs Memory.

Capacity	Admission Blocking Probability(%)
100K (bits)	62.3 ± 1.9
200K (bits)	58.4 ± 1.2
300K (bits)	58.3 ± 2.1
400K (bits)	58.2 ± 2
500K (bits)	58 ± 1.5
1M (bits)	53.6 ± 1.7
2M (bits)	46.5 ± 1.4

would approach zero. Because the connection admission process in our scheme is only related to the remaining available memory space for connection admission, more connections are admitted to the network when the memory capacity of a SS is large enough.

Figure 4.13 shows a curve (Useless Packet Rate vs. Bandwidth) when the SS's memory capacity for outgoing traffic is set to 1 M bits. A useless packet is a packet that is not transmitted on the network when its maximum delay is reached. The useless packet rate mentioned here refers to the ratio between useless bits and total bits from the four types of service flow. According to the curve in Figure 4.13, the useless packet rate decreases as the UL bandwidth assigned to a SS is increased. When the assigned UL bandwidth is 50 K bits or above, no more useless packets are produced. The result indicates that useless packets are due to a lack of UL bandwidth.

Each Y-value in Figure 13 is the mean of a sample of size 16 when the SS's memory capacity is set to 1 M bits and the assigned UL bandwidth is a specific value, the

Figure 4.13: Useless Packet Rate vs Bandwidth.

X-axis. $UPR_1, UPR_2, UPR_3, \dots, UPR_{16}$ are the values of a random sample. Each value is determined by a normally distributed random variable X with unknown expectation and unknown standard deviation. UPR_i denotes the estimation of useless packet rate at time i .

Using Equations (1), (2) and (5), we get the means, standard deviations and 90% confidence intervals (shown in Tables 4.11, 4.12 and 4.13) of useless packet rate when the SS's memory capacity is set to 1 M bits.

Table 4.11: The Means for Useless Packet Rate vs Bandwidth.

Bandwidth	Useless Packet Rate(%)
50K (bits)	41.9
100K (bits)	27.8
200K (bits)	20
500K (bits)	0
1M (bits)	0
2M (bits)	0

Figure 4.14 shows a curve (Packet Loss Rate vs. Bandwidth) when SS's memory capacity for outgoing traffic is set to 1 M bits. Different from Figure 4.9, the curve in Figure 4.14 does not consider the handling of useless packets. This figure indicates no UGS packets loss. When the bandwidth assigned to a SS is equal to 50 K bits,

Table 4.12: The Standard Deviations for Useless Packet Rate vs Bandwidth.

Bandwidth	Useless Packet Rate(%)
50K (bits)	5.4
100K (bits)	6.1
200K (bits)	0.7
500K (bits)	0
1M (bits)	0
2M (bits)	0

Table 4.13: The Confidence Intervals for Useless Packet Rate vs Bandwidth.

Bandwidth	Useless Packet Rate(%)
50K (bits)	41.9 ± 2.4
100K (bits)	27.8 ± 2.7
200K (bits)	20 ± 0.3
500K (bits)	0
1M (bits)	0
2M (bits)	0

the rtPS packet loss rate exceeds 66%, nrtPS and BE packet loss rates respectively exceed 99% and 97%. As the assigned UL bandwidth is increased, the packet loss rates decrease. When the assigned UL bandwidth is 200 K bits, the rtPS packet loss rate drops to zero, and the nrtPS packet loss rate sharply decreases to 45%. When the assigned UL bandwidth is increased to 500 K bits or above, the packet loss rates for all types of service flows decrease to zero. Comparing with the packet loss rates for rtPS service flows in Figure 4.9, the rtPS packet loss rate in Figure 4.14 is much higher. In addition, the packet loss rates for the nrtPS and BE service flows in Figure 4.14 are also higher than the ones in Figure 4.9 under the same conditions (memory capacity and UL bandwidth) when the assigned UL bandwidth is less than 500 K bits. A comparison between Figures 4.9 and 4.14 indicates that the processing of useless packets can increase the available buffer space for new incoming packets and decrease the packet loss rate. So our proposed buffer management scheme provides a QoS guarantee for multiple types of service flows by handling useless packets when the assigned UL bandwidth is insufficient.

Each Y-value in Figure 4.14 is the mean of a sample of size 16 when the SS's memory capacity is set to 1 M bits and the assigned UL bandwidth is a specific value, the X-axis. $LR_1, LR_2, LR_3, \dots, LR_{16}$ are the values of a random sample. Each value is determined by a normally distributed random variable X with unknown expectation and unknown standard deviation. LR_i denotes the estimation of packet loss rate at time i .

Figure 4.14: Packet Loss Rate vs Bandwidth (no processing useless packets).

Using Equations (1), (2) and (5), we get the means, standard deviations and 90% confidence intervals (shown in Tables 4.14, 4.15 and 4.16) of packet loss rate for the four types of service flows when the SS's memory capacity is set to 1 M bits.

Table 4.14: The Means for Packet Loss Rate (no processing useless packets).

Bandwidth	UGS LR(%)	rtPS LR(%)	nrtPS LR(%)	BE LR(%)
50K (bits)	0	66.5	99.6	97.3
100K (bits)	0	30.8	98.9	95.1
200K (bits)	0	0	44.8	90
500K (bits)	0	0	0	0
1M (bits)	0	0	0	0
2M (bits)	0	0	0	0

Table 4.15: The Standard Deviations for Packet Loss Rate (no processing useless packets).

Bandwidth	UGS LR(%)	rtPS LR(%)	nrtPS LR(%)	BE LR(%)
50K (bits)	0	3.1	0.2	1.1
100K (bits)	0	6.5	0.4	1.8
200K (bits)	0	0	5.5	2.4
500K (bits)	0	0	0	0
1M (bits)	0	0	0	0
2M (bits)	0	0	0	0

Table 4.16: The Confidence Intervals for Packet Loss Rate (no processing useless packets).

Bandwidth	UGS LR(%)	rtPS LR(%)	nrtPS LR(%)	BE LR(%)
50K (bits)	0	66.5 ± 1.4	99.6 ± 0.1	97.3 ± 0.5
100K (bits)	0	30.8 ± 2.8	98.9 ± 0.2	95.1 ± 0.8
200K (bits)	0	0	44.8 ± 2.4	90 ± 1.1
500K (bits)	0	0	0	0
1M (bits)	0	0	0	0
2M (bits)	0	0	0	0

Figures 4.15 and 4.16 show the simulation results of the pushout buffer management scheme. Figure 4.15 shows a curve (Packet Loss Rate vs. Memory) when the UL bandwidth assigned to a SS is set to 1 M bits. According to this curve, the packet loss rate for UGS service flows is always zero regardless of the memory capacity of a SS. This is because UGS service flows have the highest priority in WiMax/802.16 networks. No matter whether the buffer is full or not, an incoming UGS packet is always allowed to enter into the buffer. For rtPS, nrtPS and BE service flows, the packet loss rates decrease as the SS's memory capacity is increased. Under the same operating conditions (same memory and bandwidth), the higher the priority, the lower is the packet loss rate. The packet loss rate for rtPS service flows is the lowest. The one for nrtPS service flows is the second. The one for BE service flows is the highest. When SS's memory capacity is 50 K bits, the packet loss rate for BE service flows is about 84%, the packet loss rate for nrtPS service flows exceeds 60%, the packet loss rate for rtPS service flows is about 12%. When SS's memory capacity increases to 200 K bits or above, the packet loss rates for nrtPS and BE service flows are still higher than 58%, the packet loss rate for rtPS service flows is still higher than 10%. Although when the memory capacity increases to 2 M bits, loss of rtPS packets cannot be avoided, and the packet loss rates for nrtPS and BE service flows approach 40%. Comparing with the curves in Figure 4.11, the curves in Figure 4.15 shows much higher packet loss rates for rtPS, nrtPS and BE service flows under the same operating conditions. Due to the absence of an admission control mechanism, the pushout buffer management scheme cannot prohibit the admission of new connections into the network. In addition, the pushout buffer management scheme does not process useless packets to collect some buffer space. The pushout buffer management scheme causes such high packet loss rates that it cannot guarantee a QoS for

WiMax/802.16 service flows. It is not appropriate as a buffer management scheme for WiMax/802.16 SSs.

Figure 4.15: Packet Loss Rate vs Memory (Pushout scheme).

Each Y-value in Figure 4.15 is the mean of a sample of size 16 when the assigned UL bandwidth is set to 1 M bits and the SS's memory capacity is a specific value, the X-axis. $LR_1, LR_2, LR_3, \dots, LR_{16}$ are the values of a random sample. Each value is determined by a normally distributed random variable X with unknown expectation and unknown standard deviation. LR_i denotes the estimation of packet loss rate at time i .

Using Equations (1), (2) and (5), we get the means, standard deviations and 90% confidence intervals (shown in Tables 4.17, 4.18 and 4.19) of packet loss rate for the four types of service flows when the assigned UL bandwidth is set to 1 M bits.

Figure 4.16 shows a curve (Packet Loss Rate vs. Bandwidth) when the SS's memory capacity is set to 1 M bits. In this figure, the curve shows the UGS packet loss rate is always zero regardless of the assigned UL bandwidth. For rtPS, nrtPS and BE service flows, the packet loss rates decrease as the assigned UL bandwidth is increased. Among these three service flows, rtPS has the lowest packet loss rate.

Table 4.17: The Means for Packet Loss Rate vs Memory (Pushout scheme).

Capacity	UGS LR(%)	rtPS LR(%)	nrtPS LR(%)	BE LR(%)
50K (bits)	0	12	64	84.2
100K (bits)	0	12.1	60.1	74.6
200K (bits)	0	11.9	59.2	72.1
500K (bits)	0	8.6	54.6	58.3
1M (bits)	0	4.5	43.3	43.8
2M (bits)	0	2	39.8	40

Table 4.18: The Standard Deviations for Packet Loss Rate vs Memory (Pushout scheme).

Capacity	UGS LR(%)	rtPS LR(%)	nrtPS LR(%)	BE LR(%)
50K (bits)	0	0.1	3.1	2.3
100K (bits)	0	1	0.6	6.2
200K (bits)	0	0.6	0.1	4.5
500K (bits)	0	0.7	0.8	4.3
1M (bits)	0	1.1	1.5	5.2
2M (bits)	0	0.6	1.2	3.6

Table 4.19: The Confidence Intervals for Packet Loss Rate vs Memory (Pushout scheme).

Capacity	UGS LR(%)	rtPS LR(%)	nrtPS LR(%)	BE LR(%)
50K (bits)	0	12 ± 0.04	64 ± 1.4	84.2 ± 1
100K (bits)	0	12.1 ± 0.4	60.1 ± 0.3	74.6 ± 2.7
200K (bits)	0	11.9 ± 0.3	59.2 ± 0.04	72.1 ± 2
500K (bits)	0	8.6 ± 0.3	54.6 ± 0.4	58.3 ± 1.9
1M (bits)	0	4.5 ± 0.5	43.3 ± 0.7	43.8 ± 2.3
2M (bits)	0	2 ± 0.3	39.8 ± 0.5	40 ± 1.6

When the assigned UL bandwidth is equal to 50 K bits, the packet loss rate for BE service flows is about 70%, the packet loss rate for nrtPS service flows is about 90%, the packet loss rate for rtPS service flows is about 13%. When the assigned UL bandwidth is 50 K bits or above, the packet loss rates for nrtPS service flows sharply decreases to about 45%, the packet loss rate for rtPS service flows decreases to about 6%.

Each Y-value in Figure 4.16 is the mean of a sample of size 16 when the SS's memory capacity is set to 1 M bits and the assigned UL bandwidth is a specific value, the X-axis. $LR_1, LR_2, LR_3, \dots, LR_{16}$ are the values of a random sample. Each value is determined by a normally distributed random variable X with unknown expectation and unknown standard deviation. LR_i denotes the estimation of packet loss rate at time i .

Using Equations (1), (2) and (5), we get the means, standard deviations and 90% confidence intervals (shown in Tables 4.20, 4.21 and 4.22) of packet loss rate for the

Figure 4.16: Packet Loss Rate vs Bandwidth (Pushout scheme).

four types of service flows when the assigned UL bandwidth is set to 1 M bits.

Table 4.20: The Means for Packet Loss Rate vs Bandwidth (Pushout scheme).

Bandwidth	UGS LR(%)	rtPS LR(%)	nrtPS LR(%)	BE LR(%)
50K (bits)	0	13.7	90.4	70.3
100K (bits)	0	12.9	87.9	70
200K (bits)	0	10.2	76.5	68.8
500K (bits)	0	6.2	45.5	63.9
1M (bits)	0	3.7	34.5	46.8
2M (bits)	0	3	27.5	45.8

Table 4.21: The Standard Deviations for Packet Loss Rate vs Bandwidth (Pushout scheme).

Bandwidth	UGS LR(%)	rtPS LR(%)	nrtPS LR(%)	BE LR(%)
50K (bits)	0	1.6	5.9	3.9
100K (bits)	0	1.3	2.9	1.5
200K (bits)	0	1.9	2.5	2.9
500K (bits)	0	1.6	2.9	5
1M (bits)	0	0.7	3.5	4.3
2M (bits)	0	0.6	3.6	0.3

Comparing with the curves in Figure 4.9, the curves in Figure 4.16 show higher packet loss rates for all types of service flows when the assigned UL bandwidth exceeds 20 K bits. In addition, according to the rtPS curve in Figure 4.16, the pushout

Table 4.22: The Confidence Intervals for Packet Loss Rate vs Bandwidth (Pushout scheme).

Bandwidth	UGS LR(%)	rtPS LR(%)	nrtPS LR(%)	BE LR(%)
50K (bits)	0	13.7 ± 0.7	90.4 ± 2.6	70.3 ± 1.7
100K (bits)	0	12.9 ± 0.6	87.9 ± 1.3	$70. \pm 0.7$
200K (bits)	0	10.2 ± 0.8	76.5 ± 1.1	68.8 ± 1.3
500K (bits)	0	6.2 ± 0.7	45.5 ± 1.3	63.9 ± 2.2
1M (bits)	0	3.7 ± 0.3	34.5 ± 1.5	46.8 ± 1.9
2M (bits)	0	3 ± 0.3	27.5 ± 1.6	45.8 ± 0.1

buffer management scheme cannot avoid rtPS packets loss although the assigned UL bandwidth is increased to 2 M bits. Therefore, the pushout buffer management scheme cannot guarantee a QoS for WiMax/802.16 service flows.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this thesis, we proposed a new buffer management scheme called admission control and dynamic buffer allocation (ACDB) for outgoing traffic in a WiMax/802.16 SS. This buffer management scheme significantly decreases the packet loss rate, especially the high priority packet loss rate, by using admission control and an useless packet processing mechanisms. The packet loss rates for UGS and rtPS service flows are null, although the SS's memory capacity and assigned UL bandwidth are small. As the assigned UL bandwidth and SS's memory capacity are increased, the packet loss rates for the four types of service flows decrease. When the SS's memory capacity and assigned UL bandwidth increase to some certain values, the packet loss rates for the four types of service flows approach zero. Under the same operating conditions, the packet loss rates associated with our proposed buffer management scheme is much lower than the one associated with the pushout buffer management scheme, that has the lowest packet loss rate among the existing buffer management schemes. Because our proposed ACDB buffer management scheme can provide such a low packet loss rate, it can guarantee the QoS for the multiple types of WiMax/802.16 service flows and is suitable for being the buffer management scheme for outgoing traffic in a WiMax/802.16 SS. In addition, our simulation implementation is very closed to a real system. Therefore, we believe that our simulation program can be used, with small adaptations, as the buffer management system for outgoing traffic in a WiMax/802.16 SS.

5.2 Future Work

In our simulation, we use a round robin packet scheduling algorithm to schedule packets in buffers. This packet scheduling algorithm always serve packets according to the priority order. The higher the priority, the earlier is the service. For the same priority connections, the earlier the admission, the earlier is the service. After the packets of a connection are completely served, the immediate neighbor connection in the same connection list begins to be served. For the packets of a connection, the first in, the first served. If the current available bandwidth is not enough for transmission of a specific packet next to be served, the packet scheduling algorithm reports a lack of bandwidth and returns. It does not use that available bandwidth to transmit some packets of smaller size. When a SS is not assigned sufficient bandwidth, this kind of packet scheduling algorithm is unfair. In this case, although having the same priority, the earlier admitted connections are always served, the later admitted connections maybe not served at all. In addition, this kind of packet scheduling algorithm wastes a small amount of UL bandwidth. For instance, when available UL bandwidth for a SS is not enough to transmit a packet, but enough for the next packet in the same connection queue, the packet scheduling algorithm does not transmit the next packet although available UL bandwidth is available for it. In order to further improve the performance of our simulation, we plan to design and implement a better packet scheduling algorithm to cooperate with our proposed buffer management scheme in our future work.

Appendix A

Glossary

A-REED	adaptive random early expiration detection
ABP	admission blocking probability
AC	admission control
ACDB	admission control and dynamic buffer allocation
ADBM	adaptive dynamic input buffer management
AQM	active queue management
ATM	asynchronous transfer mode
BE	best effort
BS	base station
BW-REQ	bandwidth request
BWA	broadband wireless access
CBP	complete buffer partition
CBR	constant bit rate
CBS	complete buffer sharing
CID	connection ID
CPS	common part sublayer
CS	convergence sublayer
DBA	dynamic buffer allocation
DFPQ	deficit fair priority queue
DHCP	dynamic host configuration protocol
DL	downlink
DOCSIS	data over cable service interface specification
DPBS	dynamic partial buffer sharing
DSA	dynamic service addition
DSA-REQ	dynamic service addition request
DSC	dynamic service change
DSC-REQ	dynamic service change request
DSB	dynamic service deletion
DSD-REQ	dynamic service deletion request
FDD	frequency division duplex
FPC	fuzzy priority control
LR	loss rate
MAC	media access control
MPEG	moving picture expert group
NAR	new access router
nrtPS	non-real-time polling service
OFDM	orthogonal frequency division multiplexing
OFDMA	orthogonal frequency division multiple access
PAR	previous access router
PBS	partial buffer sharing
PDU	protocol data units
PHY	physical
PMP	point-to-multipoint
PVT	pushout with virtual thresholds
QoS	quality of service
RLC	radio link control
RSVP	resource reservation protocol
rtPS	real-time polling service
SBD	slope based discard
SDU	service data unit
SFID	service flow ID
SNMP	simple network management protocol

SP	selective pushout
SS	subscribe station
TDD	time division duplex
TDM	time division multiplexing
TDMA	time division multiple access
TFTP	trivial file transfer protocol
UBAS	uplink bandwidth allocation scheduling
UGS	unsolicited grant service
UL	uplink
UL-MAP	uplink bandwidth allocation map
UPR	useless packet rate
VBR	variable bit rate
VC	virtual connection
VoIP	voice over IP
WMANs	wireless metropolitan area networks

Bibliography

- [1] D. P. Agrawal and Q.-A. Zeng. *Introduction to Wireless and Mobile Systems*. Bill Stenquist, Department of Electrical & Computer Engineering and Computer Science, University of Cincinnati, second edition.
- [2] J. J. Alcaraz and F. Cerdan. Using buffer management in 3G radio bearers to enhance end-to-end tcp performance. In *Proceedings of the 20th International Conference on Advanced Information Networking and Applications*, pages 1–5. Department of Information Technologies and Communications, Polytechnic University of Cartagena, Plaza del Hospital, 1, 30202 Cartagena, Spain, {juan.alcaraz, fernando.cerdan}@upct.es, 2006.
- [3] G. Ascia, V. Catania, G. Ficili, and D. Panno. A fuzzy buffer management scheme for ATM and IP networks. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 2001.
- [4] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.
- [5] J. Chen, W. Jiao, and Q. Guo. An integrated QoS control architecture for IEEE 802.16 broadband wireless access systems. pages 1–6. Lucent Technologies, Bell Labs Research China, {chenjf,wjiao}@lucent.com.
- [6] J. Chen, W. Jiao, and H. Wang. A service flow management strategy for IEEE 802.16 broadband wireless access systems in TDD mode. pages 1–6. Lucent Technologies, Bell Labs Research China, {chenjf,wjiao}@lucent.com, 2005.
- [7] Y. Chen and L. Li. A random early expiration detection based buffer management algorithm for real-time traffic over wireless networks. In *Proceedings of the 2005 The Fifth International Conference on Computer and Information Technology (CIT'05)*, pages 1–5. Key Lab of Broadband Optical Fiber Transmission and Communication System Technology, University of Electronic Science and Technology of China, Chengdu, Sichuan, China, 2005.
- [8] D.-H. Cho, J.-H. Song, M.-S. Kim, and K.-J. Han. Performance analysis of the IEEE 802.16 wireless metropolitan area network. In *Proceedings of the First International Conference on Distributed Frameworks for Multimedia Applications (DFMA05)*, pages 1–8. Department of Computer Engineering, Kyungpook National University, Korea, {firecar, pimpo, kiunsen}@netopia.knu.ac.kr, kjhan@bh.knu.ac.kr, 2005.

- [9] A. K. Choudhury and E. L. Hahne. Space priority management in a shared memory ATM switch. In *IEEE*, pages 1375–1383. AT&T Bell Laboratories, Murray Hill, NJ 07974 USA, 1993.
- [10] A. K. Choudhury and E. L. Hahne. Dynamic queue length thresholds for shared-memory packet switch. In *IEEE Trans. Networking*, volume 6, pages 130–140. Bell Laboratories, Lucent Technologies, Holmdel, NJ 07733 USA, {akc, hahne}@bell-labs.com, April 1998.
- [11] L. Chuang and L. Yin. Dynamic partial buffer sharing scheme: Proportional packet loss rate. In *Proceedings of ICCT2003*, pages 259–262. Department of Computer Science and Technology, Tsinghua University, {clin, yli}@csnet1.cs.tsinghua.edu.cn.
- [12] R. Citro, S.-S. Joo, and S. Ghosh. An adaptive, dynamic buffer management (ADBM) approach for input buffers in ATM networks. In *IEEE INFOCOM 2001*, pages 2860–2864. Networking and Distributed Algorithms Lab, Department of Computer Science & Engineering, Arizona State University, Tempe, AZ 85287, 2001.
- [13] M. Crovella and A. Bestavros. *Explaining world wide web traffic self-similarity*. Computer Science Department, Boston University, October 1995.
- [14] C. Eklund, R. B. Marks, K. L. Stanwood, and S. Wang. IEEE standard 802.16: A technical overview of the wirelessman air interface for broadband wireless access. In *IEEE Communications Magazine*, pages 1–12. Roger Marks, NIST, 325 Broadway, Boulder, CO 80305, r.b.marks@ieee.org, June 2002.
- [15] R. Fan, A. Ishii, B. Mark, G. Ramamurthy, and Q. Ren. *An Optimal Buffer Management Scheme with Dynamic Thresholds*. NEC USA, Inc., C&C Research Labs, 4 Independence Way, Princeton, NJ 08540, USA, 1999.
- [16] M. Hawa and D. W. Petr. Quality of service scheduling in cable and broadband wireless access systems. In *Tenth IEEE International Workshop on Quality of Service*, pages 1–9. Information and Telecommunication Technology Center, University of Kansas, Lawrence, Kansas, 66045, {hawa, dwp}@ittc.ku.edu.
- [17] K. Helsgaun. *Discrete Event Simulation in Java*. Department of Computer Science, Roskilde University, DK-4000 Roskilde, Denmark, keld@ruc.dk, 2002.
- [18] C. S. IEEE, LAN MAN Standards Committee of the, the IEEE Microwave Theory, and T. Society. IEEE standard for local and metropolitan area networks part 16: Air interface for fixed and mobile 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 (Amendment and Corrigendum to IEEE Std 802.16-2004), May 2006.

- [19] G. Nair, J. Chou, T. Madejski, K. Perycz, D. Putzolu, and J. Sydir. IEEE 802.16 medium access control and service provisioning. In *Intel Technology Journal*, volume 8, pages 213–228, 2004.
- [20] N. Rananand. Hierarchical buffer management for ATM switches. In *IEEE*, pages 1484–1489. COMSAT Laboratories, Clarksburg, MD 20871, ranann@ntd.comsat.com, 1999.
- [21] H. Wang, B. He, and D. P. Agrawal. Admission Control and Bandwidth Allocation above Packet Level for IEEE 802.16 Wireless MAN. In *Proceedings of the 12th International Conference on Parallel and Distributed Systems (ICPADS'06)*. ORB Center for Distributed and Mobile Computing, Department of ECECS, University of Cincinnati, {wangh7, heb, dpa}@ececs.uc.edu, 2006.
- [22] K. Wongthavarawat and A. Ganz. Packet scheduling for QoS support in IEEE 802.16 broadband wireless access systems. In *International Journal of Communication Systems*, pages 81–96. Multimedia Network Laboratory, Electrical and Computer Engineering Department, University of Massachusetts, Amherst, MA 01003, U.S.A, 2003.
- [23] R.-B. Yang, Y.-S. Chu, C.-S. Wu, and M.-C. Liang. Pushout with virtual thresholds buffer management scheme in a shared buffer ATM switch. In *INTERNATIONAL JOURNAL OF NETWORK MANAGEMENT*, pages 147–154, 2003.
- [24] W.-M. Yao and Y.-C. Chen. An enhanced buffer management scheme for fast handover protocol. In *Proceedings of the 24th International Conference on Distributed Computing Systems Workshops (ICDCSW04)*, pages 1–7, 2004.
- [25] M. Zukerman, T. D. Neame, and R. G. Addie. Internet traffic modeling and future technology implications. In *Proceedings of IEEE INFOCOM*, 2003.