國 立 交 通 大 學

資訊管理研究所

博 士 論 文

UMTS 差別性服務等候機制效能應用二種佇列空間配置之研究

A Study on Performance Discussion of a UMTS Priority-based Queuing Scheme with two Queuing Buffer Allocations

研 究 生：劉芳萍

指導教授 ： 楊 千 博士

中 華 民 國 九 十 九 年 七 月

UMTS 差別性服務等候機制效能應用二種佇列空間配置之研究

研 究 生：劉芳萍　　Student：Fanpyn Liu

指導教授：楊　千　　Advisor：Chyan Yang

國 立 交 通 大 學

資 訊 管 理 研 究 所

博 士 論 文

A Dissertation

Submitted to Institute of Information Management

College of Management

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy

in

Information Management

July 2010

Hsinchu, Taiwan, the Republic of China

中華民國九十九年七月

UMTS 差別性服務等候機制效能應用二種佇列空間配置之研究

學生：劉芳萍　　指導教授:楊 千

國立交通大學資訊管理研究所

摘 要

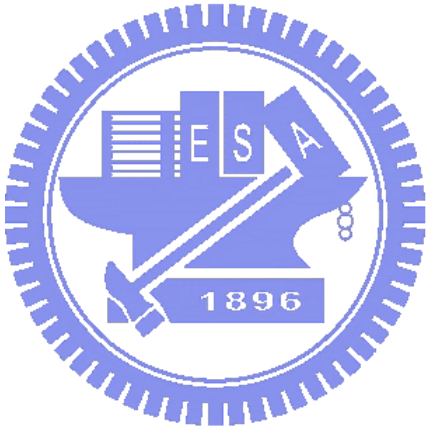
自美國、日本和歐洲各國提出第三代無線通訊系統協定標準，如 TD-SCDMA、CDMA2000 與 WCDMA / UMTS 以來，較廣為業界所看好的技術規格為 WCDMA / UMTS，因為目前使用人口最多的 GSM 歐洲系統業者也加入參與制定，且 GSM 是世界上最大的第二代移動通信網路，UMTS 勢必成為第三代行動通訊主流規格。3GPP 所提出 UMTS 的 3G 標準，可以確保 GSM 核心網路無縫 (Seamless) 接取到 3G 的核心網路，這種相容於過去 GSM / GPRS 系統的考量，對於系統業者不僅可投資延續價值，也兼顧商業上的經濟效益。

3G R5 以後的規格及 4G 網路已朝向全 IP 化的方向發展，並以 IP 作為主要通訊協定，因此，UMTS 的語音會話、影音串流、網頁互動及檔案傳輸的四種資料類型，都有其特定的屬性質，並賦予不同的封包傳輸順序，以達到符合各類型資料的服務等級 (Quality of Service, QoS) 的要求及目標，由於 IP 協定中的盡力而為服務(Best Effort Service)，對於嚴格要求低延遲、低抖動(Jitter)等服務的多媒體類或語音的資料型態，已無法滿足需求，因此，提供優化的封包轉送能力(Packet Forwarding)以達到服務等級要求，成為 UMTS 核心網路中一項重要的工作。

本論文提出的方法是在 UMTS 系統核心網路中，對不同的資料類型，提供差異性服務及優先順序的封包傳送能力，利用兩種佇列緩衝記憶體空間配置機制：動態佇列緩衝空間配置(DQB)和邏輯佇列溢流緩衝空間配置(OQB)，另外，對各類型封包允入與允出佇列時的

機制，分別以優先順序為基礎方式(Priority-Based Queuing)允入佇列及連續性加權輪詢(Sequential-Weighted Round Robin)的允出佇列機制處理。透過 NS2 網路模擬軟體，對不同方案想定、封包大小及參數值進行模擬評估，根據模擬的結果，每一種類型服務的封包在效能(Throughput)、封包丟失率(drop probability)、延遲及抖動率與 IETF 所提出之差異性服務(Differentiated Service, DiffServ)比較，兩者可達到近似的效能。

關鍵詞 UMTS、服務等級、優先佇列機制、邏輯佇列緩衝記憶體溢位、動態佇列緩衝記憶體配置、差異性服務、連續性加權輪詢佇列機制。



A Study on Performance Discussion of a UMTS Priority-based Queuing Scheme with two Queuing Buffer Allocations

Student：Fanpyn Liu Advisors：Dr. Chyan Yang

Institute of Information Management
National Chiao Tung University

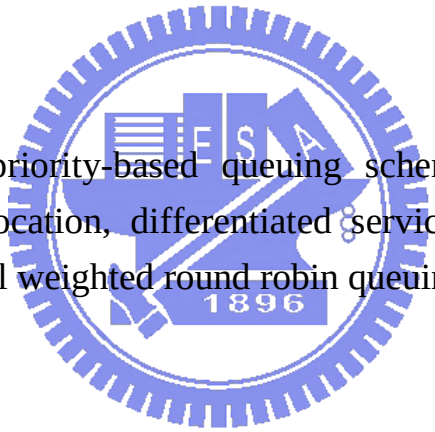
ABSTRACT

As UMTS systems will evolve into an all-IP stage in the future, packet switching becomes a prerequisite for all UMTS applications. The 3GPP defines four types of UMTS traffic: conversational, streaming, interactive and background; each type of UMTS traffic has its QoS features. Differentiated Service (DiffServ) is QoS architecture for IP networks, their based on packet marking that allows packets to be prioritized according to Service Level Agreement (SLA) management. The DiffServ is a scalable architecture and is proposed to provide QoS guarantee services and scheduling packets forwarding for each class within the IP networks.

According to four types of UMTS traffic, this study proposes a priority-based queuing scheme with two queuing buffer allocations, the dynamic queuing buffer (DQB) allocation and the overflow queuing buffer (OQB) allocation, to support packet forwarding of four types of UMTS traffic in a DiffServ method. In the proposed queuing scheme, two major modules, a priority-based enqueuing module and a sequential weighted round robin (SWRR) dequeuing module, base on the DQB

allocation and the OQB allocation to perform differentiated packet enqueuing / dequeuing jobs among four types of UMTS traffic.

In this study, we use the ns2 (Network Simulator version 2) as the simulation platform to simulate several scenarios. Discussing the simulation results and analysis, we can find the proposed queuing scheme can base on packet transmission priorities of four types of UMTS traffic to support a differentiated packet forwarding behavior among UMTS traffic in a UMTS core network and the performance of UMTS traffic with a high priority always gets a better packet forwarding performance than that of UMTS traffic with a low priority. And, the differentiated packet forwarding behaviors with the proposed queueing scheme are similar to the packet forwarding behavior with the IETF DiffServ scheme.



Keyword: UMTS, priority-based queuing scheme, DQB allocation, OQB allocation, differentiated service packet forwarding, sequential weighted round robin queuing scheme.

誌 謝

要開始寫「誌謝」這段，代表博士班的學生生涯要告一段落了，沒有胸懷壯志，也沒有驪歌高唱，只有一路走來受到師長照顧和幫忙，感激的心情無以言表，想說的感謝話有很多很多，想表達的誌謝辭也很長很長，遠遠超過這本論文的長度，非三言兩語可以道盡的。

能完成這個學位，首先要感謝的人，是我的指導教授楊 千博士，因為教授有教無類，願意收我為徒成為大師的門下，才得以一窺學術領域的深奧、作研究的嚴謹和人生智慧的哲理，生活中教授對於嚴肅的主題，常以詼諧幽默和逗趣的方式闡述其中道理，並也常不經意的在四兩撥千金的談笑間，剖析事情的角度分析其中的智慧，令我們佩服、震撼且感動不已。另外，我還要特別感謝的人是傅振華學長，因為他孜孜不倦、耐心和包容的指導，論文寫作、投稿甚至程式的撰寫、除錯和修改...等，才使得研究成果得以展現，完成這本論文的撰寫，心中有太多的感謝想要表達，千言萬語亦不足以道盡，能敘述得出來的也不及千萬分之一。

徐道義教授在我就學期間，仍無時無刻關心我的研究狀況，並指導我作研究的技巧與態度，此外，也要感謝口試委員－羅濟群教授、劉敦仁教授和吳宗禮教授們逐字斧正論文內容，並犧牲週末假期參加學生計畫書及論文口試，並於論文修改期間不吝地提供我許多寶貴意見，啟發學生對問題的深究獲益良多，使本篇論文才能更加充實完善。

另外，實驗室的學弟妹們－耿杰、建全、良駒、憲明、阿國、秋皓、士原和志棋...等一起通宵達旦作研究、討論、吃宵夜、打球、閒聊八卦及搞笑...等的革命情感，還有同班同學姐妹淘－翠娟、秀怡和明賢，以及俊龍和文茂平大哥在心情沮喪研究低潮時，相互勉勵加油打氣一起出遊、唱卡啦 OK 瘋狂的同窗之誼，這些都是我千年修來的福分，才能和他們成為同學；另外在我職場中的長官、同事也都是背後推動我前進堅持下去的動力。

最後，謹以此文獻給我摯愛的雙親、小弟和懷念的大弟博熙及愛犬(胖胖柴)，由於有他們默默的支持和陪伴，讓我能撐過這些煎熬的日子，年邁的父親期盼和我分享這份榮耀的日子已經很久了，感謝他至始至終對我的寬容和信心，並也要感謝我的愛犬們(蜜蜜和PP)，因為牠們總是很有耐心靜靜的，陪我渡過一天又一天熬夜 K 書、寫論文的日子。

要感謝的人及事有太多族繁不及備載，一路走來受到太多人的扶持和幫助，在博士生涯的研究歲月裏，也僅知學問與研究之不易，如訪隱者之不遇，時時喟嘆「只在此山中，雲深不知處」，知識的累積尚欠火候，回首這些年來，雖然其中沒有什麼值得特別炫耀的成果，但對我而言，是寶貴的人生歷練，是無數教誨、關愛和幫助的結果，本論文的完成絕非終點，文中的不足和淺顯之處則是我新的征途上，另一個負笈研究的新起點，感謝大家。

Contents

List of Tables

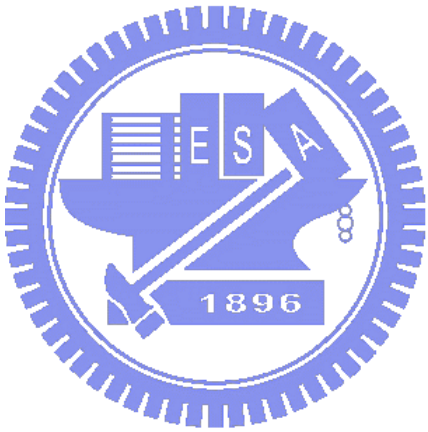
List of Figures

Abbreviations

3G	3rd Generation
4G	4th Generation
3GPP	3rd Generation Partnership Project
APD	Average Packet Delay
APJ	Average Packet Jitter
AQM	Active Queue Management
BAC	Background traffic
BS	Base Station
CDMA	Code Division Multiple Access
CN	Core Network
CON	Conversational traffic
DiffServ	Differentiated Services
DQB	Dynamic Queueing Buffer
DP	Dequeued Packets
DPR	Dequeued Packet Ratio
DPT	Disorder Packet Transmission
DRP	Dropped Packets
DRPR	Dropped Packet Ratio
ECN	Explicit Congestion Notification
EDGE	Enhanced Data Rates for GSM Evolution
EI	Evaluation Item
FDD	Frequency Division Duplex
FIFO	First In First Out
FTP	File Transfer Protocol
GBS	Guaranteed Buffer Size
GGSN	Gateway GPRS Support Node
GoS	Grade of Service
GPRS	General Packet Radio Service
GSM	Global System for Mobile communication

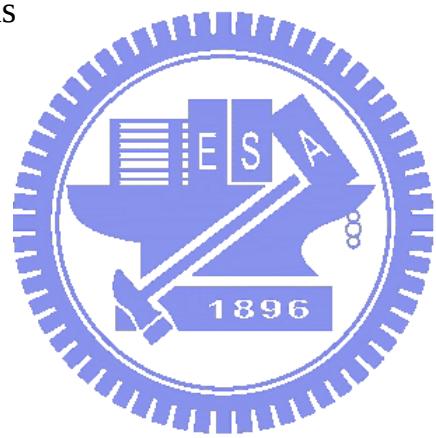
IETF	the Internet Engineering Task Force
INT	Interactive traffic
IntServ	Integrated Service
IP	Internet Protocol
IPTV	Internet Protocol Television
MT	Mobile Termination
NS2	Network Simulator 2
OQB	Overflow Queueing Buffer
PDTT	Packet Dequeueing Turn Transfer
PEP	Packet Enqueueing Probability
PHB	Per-Hop-Behavior
PKT	Packet
PQM	Passive Queue Management
PRI	Priority Queuing
PS	Packet Size
QoS	Quality of Service
RED	Random Early Detection
RFC	Request for Comment
RNC	Radio Network Controller
RR	Round Robin
SAP	Service Access Point
SDU	Service Data Unit
SLA	Service Level Agreement
SMS	Short Messaging Service
STR	Streaming traffic
SWRR	Sequential-WRR (Weighted Round Robin)
TCL	Tool Command Language
TCP	Transport Control Protocol
TDD	Time Division Duplex
TE	Terminal Equipment
TD-SCDMA	Time Division Synchronous Code Division Multiple Access

TPFPR	Total Packet Forwarding Performance Ranking
TT	Transmission Time
UDP	User Datagram Protocol
UE	User Equipment
UMTS	Universal Mobile Telecommunications System
UT	UMTS Traffic
UTRA	UMTS Terrestrial Radio Access
UTRAN	UMTS Terrestrial Radio Access Network
VoIP	Voice over Internet Protocol
WCDMA	Wireless Code Division Multiple Access
WRR	Weighted Round Robin
WS	Weight Sum



Symbol Description

B	Bytes
D(i, j)	The jitter between any two particular packets.
i or j	Send and receive times between two packets(i and j)
ms	10^{-3} seconds
R_i	The arrival time of packet i
R_j	The arrival time of packet j
S_i	The sending time of packet i
S_j	The sending time of packet j
μs	10^{-6} seconds



Chapter 1. Introduction

Nowadays, mobile communications have become popular communication fashions worldwide and are available to all. The evolution technologies over the last two decades has enabled the development of the ubiquitous mobile communication service, which can provide the mobile user with voice, data and multimedia services at any time, any place, and in any format. Hence, many applications and bandwidth requirements are proposed for mobile communications. The third generation (3G) and the proposed fourth generation (4G) mobile communication support a broadband mobile communication environment and diversified services.

1.1. Research Background and Motivation

The Universal Mobile Telecommunications System (UMTS), one of the 3G mobile communication standards developed in Europe, supports diversified mobile communication applications [1, 2]. According to 3GPP planning, an all IP-based architecture will eventually be adopted in the UMTS core network to support diversified 3G services [3]. For reducing the costs to increase the revenue, the network service providers plan to merge the data communication network and telephony communication network and develop the all-IP networks. In an all-IP network, traffic is packetized and transmitted within a UMTS core network and external IP networks [4-7]. These traffic types of UMTS applications can be divided into four classes: conversational, streaming, interactive, and background. For 3G applications, the 3GPP defines four types of traffic, each with different Quality of Service (QoS) features. Differentiated services (DiffServ) must be supported within a UMTS core network to satisfy the required QoS of UMTS applications [8, 9]. The user will be judged on the basis of the these applications, on the other hand, network bandwidth and application priority will determine

the performance of the network under its control in terms of the cost of operating the network to support the agreed upon QoS requirements for its users.

Today, people expect their mobile communication service to be available all the time with no deterioration in the service quality. Hence, it is imperative that the evaluation of the performance of the mobile network takes into account the survivability requirements. With a QoS solution based on different QoS classes the use of the mobile network resources can be optimized. The users of the new networks services are only interested in end-to-end QoS [10]. End-to-end services typically involve communication through external networks, which make it obligatory to be able to map UMTS QoS parameters to external network QoS parameters and vice versa.

The main goal of this study is to analyze and compare the performance of different queuing scheme over UMTS core network gateway, using two queueing buffer allocations. We consider four classes of packets which have to be served, where every packet of class has priority forwarding behavior corresponding to the QoS requirement. The simulation results are collected and analyzed to understand the performance of the proposed queuing schemes. The priority-based queuing scheme is implemented using an ns2 simulator, through which several scenarios are assumed and simulated. Finally, a conclusion is provided.

1.2. Related Research Work

In the competitive communication world, network bandwidth is a precious and limited resource. As with any mobile communication network service, guaranteed service levels and network performance are critical factors. We investigate the queue scheduling scheme for the provision of QoS over UMTS core network gateway by network

performance. In fact network performance should not be measured in absolute quantities like dropped packets but the degree to which the network satisfies the service requirements of each application. Critical network parameters and performance to be used in specifying and assessing the speed, accuracy, dependability, and availability of IP pack transfer [11]. These parameters include packet drop, transfer delay, packet jitter, and throughput. In particular, the network performance and reliability are embodied by these QoS parameters. In order to reach the goal, the access and delivery rules have to be formalized and be able to make meaningful promises in end-to-end QoS.

To realize satisfying QoS results, investigating the process of UMTS performance in a form of network simulation is a required in implementing a particular queuing discipline. The simulation can evaluate parameters associated with UMTS network performance; delay, packet loss, packet jitter and throughput in accessing new service and architecture. Packet loss information can be useful in tracking persistent congestion problem. The statistical characteristics of the lost packets are based on established loss models such as Gilbert, Poisson, and Bernoulli [12, 13]. The probability of packet loss resulted from each of sent packets. That is:

$$Packet\ Loss\ Ratio = \frac{Number\_of\_Lost\_Packets}{Total\_Transmitted\_Packets} \qquad \ldots\ldots\ldots \quad (1)$$

Jitter is defined as the mean deviation of the difference in packet spacing at the receiver compared to the packet spacing at the sender for a pair of packets. This value is equivalent to the deviation in transit time for a pair of packets. The jitter metric is defined as the difference in send and receive times between two packets, *i* and *j*. The difference, *D(i, j)*, provides the jitter between any two particular packets, however, a jitter value which measures the accumulated jitter over all packets is required.

$$D(i, j) = (R_j - S_j) - (R_i - S_i) \quad \ldots\ldots\ldots \quad (2)$$

where S_i*=the sending time of packet i;* R_i*=the arrival time of packet i;*
Sj*=the sending time of packet j;* R_j*=the arrival time of packet j*
and $j > i$

The jitter of the response time is very important for real-time applications such as telephony. Web browsing and mail are fairly resistant to jitter, but any kind of streaming media (voice, video, music) is quite susceptible to jitter. Jitter is a symptom that there is congestion, or not enough bandwidth to handle the traffic.

1.3. Organization of the Dissertation

This dissertation is organized as follows. Related research works and literature review are surveyed in Chapter 2. Chapter 3 introduces some queue management mechanisms related to construct a queueing scheme. In Chapter 4 and Chapter 5, they are the most important core of this dissertation. Here we describe our queueing module and priority based queueing scheme with two different queueing buffer allocations in a UMTS core network gateway. Then, chapter 5 presents some simulation results to compare performance of priority-based queueing scheme with two queueing buffer allocation. Finally, we conclude our research and summaries in Chapter 6.

Chapter 2. Literature Review

2.1. UMTS Services and Applications

UMTS is a 3G broadband, packet-based transmission of text, digitized voice, video, video conferencing, IPTV (Internet Protocol television) and multimedia at data rates up to 2Mbps, can be reached. Higher bit rates naturally facilitate some new services, such as video telephony and quick downloading of data [14, 15]. If there is to be a killer application, it is most likely to be quick access to information and its filtering appropriate to the location of a user. Often the requested information is on the Internet, which calls for effective handing of TCP/UDP/IP traffic in the UMTS network. At the start of the UMTS era almost all traffic will be voice, but later the share of data will increase. It is, however, difficult to predict the pace at which the share of data will start to dominate the overall traffic volume. At the same time that transition from voice to data occurs, traffic will move from circuit-switched connections to packet-switched connections. At the start of UMTS service not all of the QoS functions will be implemented and therefore delay-critical applications such as speech and video telephony will be carried on circuit-switched bearers. Later, it will be possible to support delay-critical services as packet data with QoS functions [16].

Compared to GSM and other existing mobile networks, UMTS provides a new and important feature, namely it allows negotiation of the properties of a radio bearer. Attributes that define the characteristics of the transfer may include throughput, transfer delay and data error rate. To be a successful system, UMTS has to support a wide range of applications that possess different QoS requirements. At present it is not possible to predict the nature and usage of many of these applications. Therefore it is neither possible nor sensible and optimise UMTS to only one set of applications. Table 1 illustrates how the applications enabling 3G service categories [17].

Table 1. Applications Enabling 3G Service Categories

Service Categories / Application	*Location Based*	*Edutainment and Infotainment*	*B2C Service*	*Office Extension*	*Tele-medicine*	*Telematics Telemetry Monitoring*
Multimedia	H	H	H	H	M	L
m-commerce	H	H	H	L	L	M
Unified messaging	L	L	L	H	M	L
VoIP	M	M	M	H	H	L
Interactive Broadcasting	H	H	M	L	L	L
IP Access	M	H	H	H	M	H
Positioning	H	H	L	M	L	H
Legends: **L:** Low importance, **M:** Medium importance**, H:** High importance						

2.2. Requirements for QoS

3GPP has specified high level requirements for UMTS QoS. These requirements are divided into three categories – end user, general and technical requirements. According to the 3GPP planning, an all IP-based architecture will be adopted in the UMTS core network eventually to support diversified 3G services. In an all-IP network, traffic is packetized and transmitted within a UMTS core network and external IP networks [3]. For 3G applications, the 3GPP defines four types of traffic; their QoS features are different. DiffServs must be supported within a UMTS core network to satisfy the required QoS of UMTS applications [18]. The performance of packet forwarding process is one of the important factors that affect QoS of UMTS applications. For UMTS packet forwarding process in a UMTS core network gateway, a queuing scheme always plays an important role. A proper priority-based queuing scheme within a UMTS core network can support packet forwarding in a DiffServ way for UMTS applications.

This study focuses on performance comparison of priority-based queuing scheme with two different queuing buffer allocations in a

UMTS core network gateway. The proposed queuing scheme is implemented in an ns2 simulator and several scenarios are assumed and simulated. The simulation results are collected and analyzed to observe the performance of the proposed queuing scheme with different queuing buffer allocations. Finally, a conclusion is reached.

2.2.1. QoS Mechanisms

QoS is one of the most important issues in networks in general, and particularly so in the Internet and other IP networks. QoS deals with the strict management of traffic such that guarantees can be made and SLAs (Service Level Agreement) between customers and service providers can be observed. In the case of packet switching, QoS basically guarantees that a packet will travel successfully between any two points. In packet-switched networks, these parameters need to be controlled in order to guarantee QoS, including latency end to end, jitter, loss, sequencing (i.e., the order of delivery of the packets), and errors etc..

Figure 1 shows the four main IP QoS mechanisms: classification (used for packet identification), conditioning (used for traffic shaping), queue management (used to manage the queue depth), and queue scheduling (used for packet scheduling) [19].

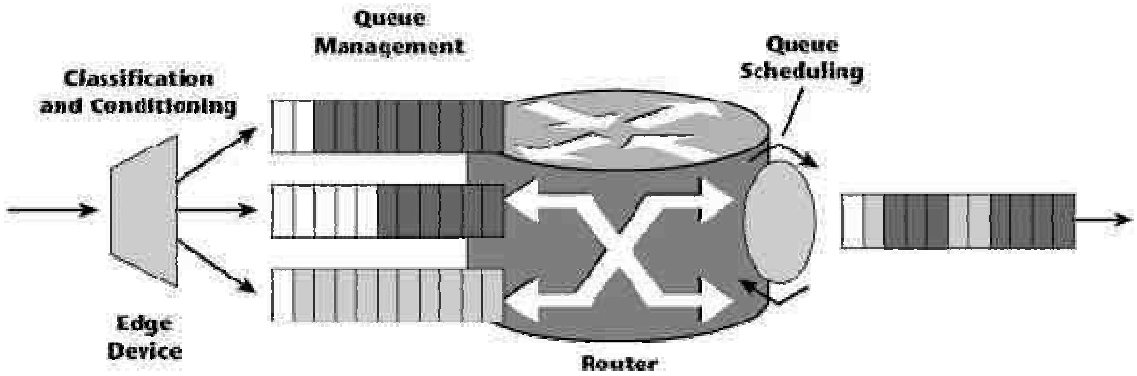


Figure 1. IP QoS mechanisms

When implementing QoS, a common mechanism used is queuing, We survey a variety of queuing strategies that manage resources where congestion might occur. In UMTS networks, the transition from a TE (Terminal Equipment, such as mobile phone, Laptop…etc) via RNC (Radio Network Controller) to a UMTS core network makes the gateway in between a congestion point.

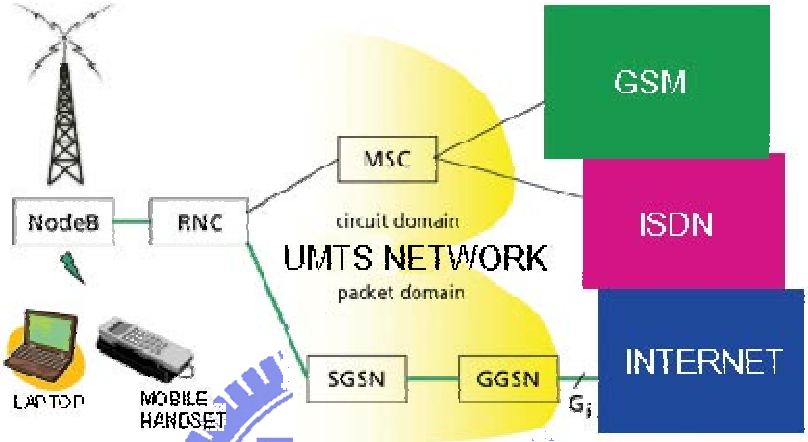


Figure 2. Diagram of the infrastructure of a UMTS network

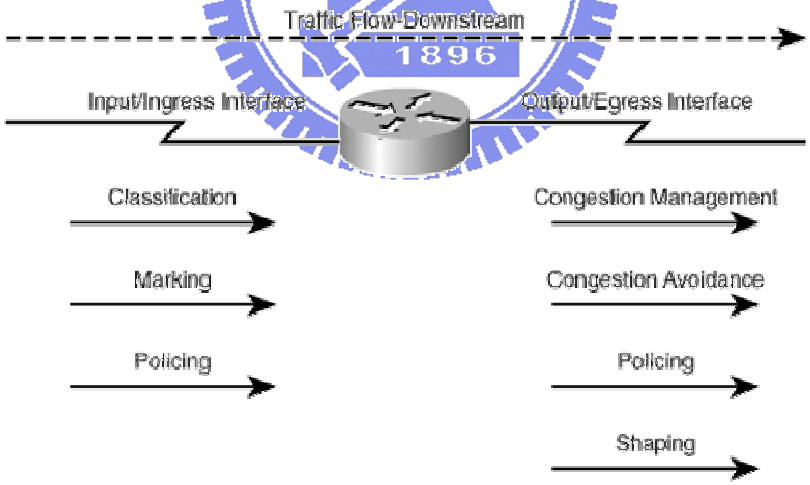


Figure 3. The QoS functions and can be implemented on a router

In such cases, queuing might be configured on the gateway at the network edge. Figure 2 shows diagram of the infrastructure of a UMTS network [2]. Figure 3 illustrates the QoS mechanisms can be implemented the UMTS core network and briefly describes the QoS functions [19].

2.2.2. UMTS QoS Architecture

Network services are considered end-to-end, this means from a Terminal Equipment (TE) to another TE. An End-to-End service may have a certain QoS which is provided for the user of a network service. It is the user that decides whether he is satisfied with the provided QoS or not. To realize a certain network QoS a Bearer Service with clearly defined characteristics and functionality is to be set up from the source to the destination of a service. A bearer service includes all aspects to enable the provision of a contracted QoS. These aspects are among others the control signaling, user plane transport and QoS management functionality. A UMTS bearer service layered architecture is depicted in Figure 4, each bearer service on a specific layer offers its individual services using services provides by the layers below [10].

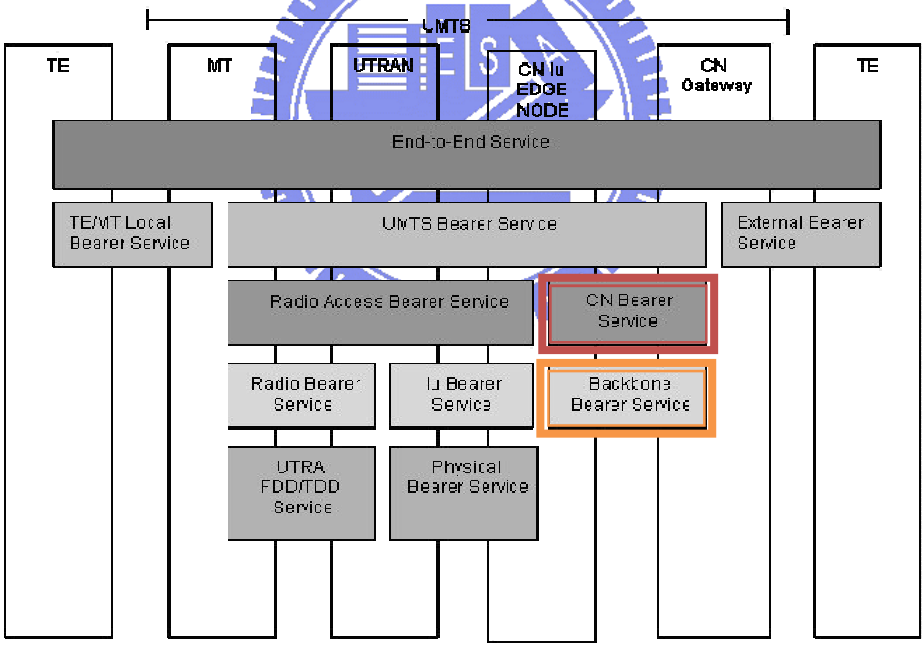


Figure 4. UMTS QoS Architecture

Legends:
TE: Terminal Equipment, **MT**: Mobile Termination, **CN**: Core Network,
UTRAN: UMTS Terrestrial Radio Access Network,
EDGE: Enhanced Data Rates for GSM Evolution,
lu: the lu interface to connect UTRAN and CN
UTRA: Universal Terrestrial Radio Access
FDD: Frequency Division Duplex
TDD: Time Division Duplex

2.2.3. QoS Classes of UMTS applications

According to UMTS applications' features, four QoS types of traffic, conversational, streaming, interactive, and background, are defined by the 3GPP. Each type of traffic has its QoS features. Four types of UMTS traffic are described as follows [10].

- **Conversational Traffic**

VoIP, videoconference, and video telephony are the typical conversational traffic. Real time is the most important characteristic for conversational traffic. Moreover, to support conversational traffic, a low delay and low jitter service is required. The transfer delay will be significantly lower than the round trip delay of an interactive application. The acceptable packet transfer delay for conversational traffic is very stringent.

- **Streaming traffic**

Watching a real time video or listening to a real time audio from a video/audio server through the UMTS network is a typical streaming application. Generally speaking, streaming traffic is one way packet transport, and packets are transmitted to users in real time. Streaming traffic require a small delay variation. However, low transfer delay is not required.

- **Interactive traffic**

When an end user accesses information or data from a server through a UMTS network, it belongs to interactive traffic. Typical interactive UMTS applications include web browsing, chatting room, ICQ, and telnet; they require low packet loss rate. A reasonable packet transfer delay is allowed for interactive traffic. In general, interactive UMTS applications are classical data communication applications that are characterized by the request/response pattern of the end-user. Round trip delay time is one of the key attributes. A low bit error rate of

transmitted packets is another characteristic.

- **Background traffic**

Background delivery of E-mails, SMS, FTP, and reception of measurement records within a UMTS network are the typical background traffics. Background traffic transmission is also one of the classical data communication schemes that do not expect the data to be reached within a certain period of time. The QoS of background traffic requires a low packet loss rate and relaxed delay requirements, similar to the transmission of best effort traffic. Therefore, packet transfer of background traffic is more or less time insensitive with a low bit error rate.

Table 2. UMTS QoS features

Traffic class	***Fundamental features***	***Application example***	***Packet transmission priority***
Conversational	- Preserve time relation between information entitles of the stream -Sensitive to packet delay -Conversational pattern	VoIP, Video telephony	1st
Streaming	- Preserve time relation between information entities of the stream - Sensitive to packet delay	Streaming video, Streaming audio	2nd
Interactive	-Request/Response pattern - Preserve payload content -Low error packet rate	Telnet, Chartroom, Web browser	3rd
Background	-Destination is not expecting the-data within a certain time -Preserve payload content, -Insensitive to packet delay, -Low error packet rate	E-mail, SMS, File transfer	4th

From the QoS feature descriptions of UMTS traffic that mentioned above, the major distinguishing factor among these QoS features is the time sensitivity of packet transfer delay. Conversational traffic is the most delay sensitive, followed by streaming and interactive applications. Background traffic is the least delay sensitive. Moreover, compared to conversational and streaming traffic, interactive and background traffic

is sensitive to packet error rate since most of them are traditional Internet applications. In addition, interactive traffic bases on a request/response operation pattern, a long packet delay is not allowed. Thus, interactive traffic reaches a higher packet transmission priority than background traffic. The QoS features of UMTS traffic are summarized in Table 2 [16].

2.3. UMTS with Differentiated Services (DiffServ)

The UMTS has a number of service classes that require end-to-end QoS support. This requirement imposes on the design of the UMTS core network bearer service as a part of the UMTS hierarchical QoS architecture. This study presents queuing scheme and some scenario simulations for provisioning QoS in the UMTS core network based on the DiffServ model, a relatively simple but scalable IP-based QoS technology. This requires proper choices of QoS mapping, router configurations, and queuing scheme. Efficient queuing schemes are introduced for the UMTS core network gateway, particularly for the scheduling and buffer management schemes, to enhance QoS provisioning in UMTS. The effectiveness of this approach is illustrated by computer simulations.

The next generation of mobile phones will be probably all-IP based enabling users to access Internet services. In order to make this possible a satisfactory QoS, at least equal to the fixed Internet, must be ensured. To achieve this goal an end-to-end QoS system must be constructed. Another fact is the dominance of IP over other technologies due, it is important to develop end-to-end IP QoS guarantees for the different applications over distinct access technologies [10]. This is particularly important for cellular wireless networks due to the ever growing expansion of mobile phone users. One way to contribute to this goal is to apply DiffServ QoS mechanisms to UMTS technology in order to model

an End-to-End QoS communication system. In particular, RED (Random Early Detection) queue management and PRI (Priority Queuing) or WRR (Weighted Round Robin Queueing) scheduling policies are enforced [20]. Different UMTS traffic classes are mapped into different DiffServ parameters [21, 22]. The performance of this architecture has been evaluated by simulation using NS2, assuming different network load scenarios.

2.3.1. Integrated Services (IntServ)

The integrated services (IntServ) scheme focuses on end-to-end individual packet flows. In this scheme is designed to provide a set of extensions to best-effort service model. The service level can be typically quantified as a minimum service rate, or a maximum tolerable end-to-end delay or loss rate. According to available resources, the network grants or rejects the flow requests. The admission control unit, the packet forwarding mechanisms and the Resource Reservation Protocol (RSVP) are the three major components of the IntServ architecture; these three components perform resources availability check, packet forwarding process in a router and bandwidth reservation jobs [23].

Since flow-based integrated services are supported in the IntServ scheme; routers require more complicated mechanisms to maintain control and packet forwarding states of all flows passing through them. For a router, it is a heavy load. From an viewpoint of implementation and operation, it is a significantly difficult job. Therefore, there exist scalability and manageability issues for the IntServ scheme [24]. The design principles of RSVP provides the end-to-end QoS guarantee for data stream and it is to reserve and maintain the resources at each node that is in the transmission path of flows. Figure 5 illustrates the best effort service and the IntServ. In this illustration, the term "traffic flow" is used in a loose sense and represents the source of traffic. In the best

effort service, all packets are lumped into a single mass regardless of the source of the traffic. In IntServ, individual flows are distinguished on an end-to-end basis.

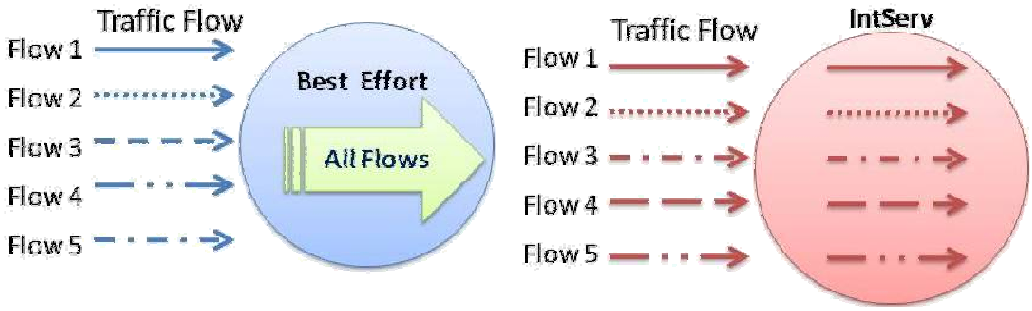


Figure 5. Sketch map of Best effort and IntServ

2.3.2. Differentiated Services (DiffServ)

DiffServ attempts to accomplish the same goal as IntServ with better scalability, it is a simple model where traffic entering a network is first classified and possibly conditioned at the boundaries of the network, and then assigned to different behavior aggregates. Scalability is improved by moving per-flow states to edge routers and keeping information of only a few flow classes called Per-Hop-Behavior (PHB) in the core gateways. Packets are treated according to their PHB classes instead of individual flows. PHB can be performed by proper buffer management mechanisms such as RED in the gateways and fulfills scalability by performing all complicated QoS function, such as traffic classification, marking, metering, conditioning, shaping and per-flow traffic [22, 25].

The Diffserv control architecture definition shows in Figure 6, these functional elements are located in the ingress node of a DiffServ domain and in interior DiffServ-compliant nodes [26].

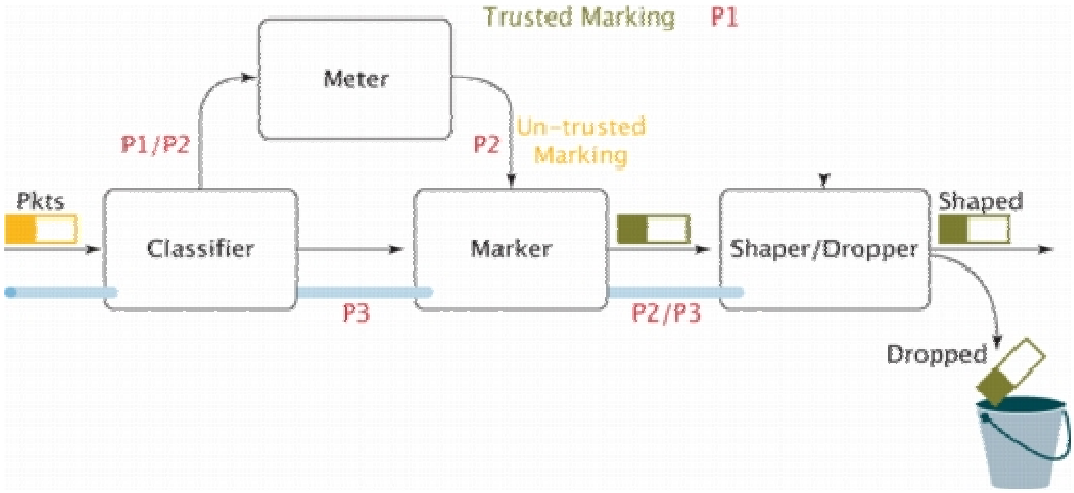


Figure 6. DiffServ Control Architecture

Classifier: Selects a packet in a traffic stream based on the content of some portion of the packet header.

Meter: Checks compliance to traffic parameters (ie: token bucket) and passes results to the marker and shaper/dropper to trigger action for in/out-of-profile packets.

Marker: Writes/rewrites the DSCP value

Shaper: Delays some packets to be compliant with the profile.

The DiffServ has no dynamic admission control. Therefore, the network managers must make sure that enough resources are available for the agreed SLAs. DiffServ doesn't support per-flow QoS guarantees to achieve scalability. It becomes challenging to still maintain QoS, especially for voice and video, which need per-flow guarantees [27]. The ways in which PHBs, edge functionality, and traffic profiles can be combined to provide an end-to-end service, such as a virtual leased line service [28].

Chapter 3. Review of Queue Management Mechanisms

The desired performance level and applying resource allocation policies in mobile communication network is a hard task. There is a need to express policies defined by the queuing scheme of each resource in a UMTS core network gateway. This need is becoming increasingly pressing in settings such as minimum limit, maximum limit, buffer size, type of packet, packet enqueueing probability…etc. The problem of queuing scheme in mobile communication network is becoming an important one.

3.1 Passive Queue Management (PQM)

In passive queue management (PQM), packets coming to a buffer are rejected only if there is no space in the buffer to store them and the senders have no earlier warning on the danger of growing congestion [29]. In this case all packets coming during saturation of the buffer are lost. The existing schemes may differ on the choice of packet to be deleted (end of the tail, head of the tail, random). During a saturation period all connections are affected and all reach in the same way, hence they become synchronized.

The main drawbacks of PQM are summarized as follows [29].

- **Global synchronization**

When a drop-tail buffer is full, all of the incoming packets are dropped. Consequently, all the affected TCP connections try to recover those dropped packets at about the same time. This moment, all the connections simultaneously send large amount of packets to congest the buffer again. This phenomenon may seriously affect the link utilization and thus the overall network performance.

- **Full-queue**

PQM is only activated after the buffer is full. The buffer occupancy may oscillate between empty and fullness. A traffic flow may experience large end-to-end variations.

- **Lock-out**

Because of global synchronization phenomenon, some connections are always served first and the others are denied by PQM. The network resources are thus not distributed in a fair manner. This is called “lock-out”.

3.2 Active Queue Management (AQM)

To enhance the throughput and fairness of the link sharing, also to eliminate the synchronization, the Internet Engineering Task Force (IETF) recommends active algorithm of buffer management [30]. Active queue disciplines drop or mark packets before the queue is full. Typically, they operate by maintaining one or more drop/mark probabilities, and probabilistically dropping or marking packets even when the queue is short. Active Queue Management (AQM) can improve the performance of TCP, and has been recommended by the IETF for use in the routers of the mobile communication [29]. The goal of AQM is three folds. First, to improve throughput by reducing the number of packets dropped. This is achieved by keeping the average queue length small in order to absorb naturally occurring bursts without dropping packets. Second, AQM provides low delay to interactive services by maintaining a small average queue length. Third, AQM avoids the lock out phenomenon arising from tail drop [31, 32].

3.3. Random Early Detection (RED)

Among various active queue management schemes, RED is probably the most popular studied. RED was proposed to improve the performance of TCP connections. As a queue management mechanism, it drops packets in the considered gateway buffer to adjust the network traffic behavior according to the queue size. Clearly, the configurable parameters of RED such as dropping probability and thresholds are critical to network performance, but the choice of these parameters remains more of an art than a science because of the complexity of the relationship between TCP/RED parameters and network performance. A few papers on mathematical models and parameter settings can be found in the literature [33, 34].

RED is an effective mechanism to control the congestion in the network routers. It also helps prevent the global synchronization in the TCP connections sharing a congested router and to reduce the bias against burst connections. It was an improvement over the previous proposals, such as Random Drop and Early Random Drop. Clark and Fang [35] have proposed the incorporation of DiffServ in the Internet by applying RED with different parameter setting to the "In" and "Out" packets of the flows arriving at a router. To be able to apply RED mechanism in the DiffServ service, it is important to survey its queueing behavior. For example, to figure out any guarantees on throughput and delays one needs to survey these as a function of the RED parameters. It is useful to complement these efforts with an analytical study.

Lin and Morris [36] have shown that the RED scheme doesn't work particularly well when the queue is occupied by well-behaved TCP flows as well as greedy UDP flows at the same time. Misbehaving flows don't back off even if their packets are dropped. The average value of the queue remains over Min_{th}, causing drop from TCP flows that have already reduced their rate.

3.4. Priority-based Queueing

Priority-based queueing is a simple approach to provide DiffServ to different packet flow. Packets of different flows are assigned a priority level according to their QoS requirement. When packets arrive at the output link, they are first classified into different classes enqueued separately based on their priorities. Then, queues are served in order. The highest priority queue is served first before serving lower priority queues. Packets in the same priority class are serviced in a FIFO manner. But if a higher priority packet arrives while serving a lower priority packet, the server waits until complete the service of the current packet then goes back to serve the higher priority queue [37, 38].

Accordingly, higher priority queues will always be served before lower priority queues. If a high priority user offers more load than the link capacity of the output link, no packets can be transmitted from the lower priority queues. In the worst-case, all packets in lower priority queues will be discarded due to exceeding the transit delay bounds of the scheduler. The high-priority queue is always going to be given first priority, and that means traffic in the lower-priority queues can sit there for a long time. Figure 7 illustrates the priority-based queue, anytime a packet enters the high queue, the scheduler will stop transmitting any other queue's traffic and transmit the high-priority traffic. The packets in the other queues now have to wait their turn, if too many packets enter those higher-priority queues. The starvation or blocking problem always arises with high occurrence probability in the lower priority queue.

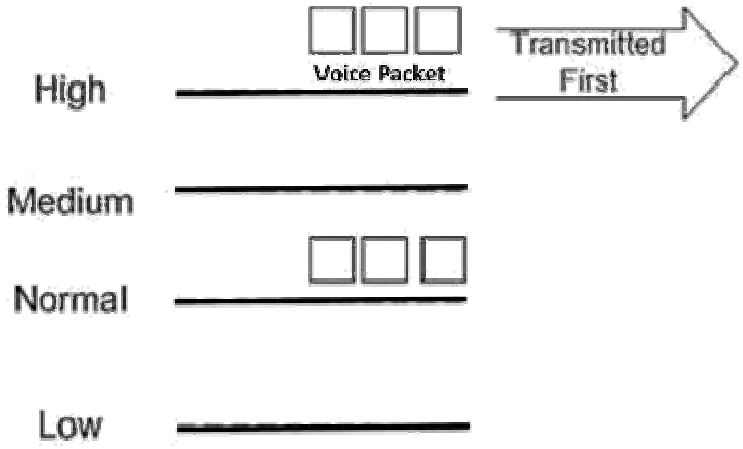


Figure 7. Priority-based Queueing

The priority-based queuing discipline play a crucial role in the implementation of the DiffServ architecture where packets are classified into a number of traffic classes and handle with various priorities. Numberous research efforts have been made on performance analysis and evaluation of the priority queuing mechanism [29, 39], as well as its development and applications [40, 41].

3.5. Weighted Round Robin (WRR) Queueing

The traditional first-in-first-out (FIFO) droptail was initially the only queue management scheme in the network. It is simple and easy to implement in routers, however, it exacerbates the limitations of end-point congestion control schemes such as TCP. When a packet arrives and the queue is currently full, the incoming packet will be dropped. Droptail is the most widely used queue manage algorithm due to its simple implementation and relatively high efficiency. However, droptail has some weakness, such as the bad fairness sharing among TCP connection, and the throughput and link efficiency suffer severe degradation if congestion is getting worse [42]. Figure 8 and Figure 9 illustrate FIFO queueing and WRR diagrams.

In WRR assigns a weight to each queue, and it then services each nonempty queue in proportion to its weight, in round-robin fashion. The packets in different queues are processed in turn. Thus the lowest-priority queue can be guaranteed of a minimum bandwidth. This avoids the case that the packets in the low priority queues cannot be served. WRR is optimal when using uniform packet sizes, a small number of flows, and long connections [43]. Lindemann and Thummler [44] have shown a QoS differentiation with WRR, but the main focus in the paper has been in balancing the resources between circuit switched voice calls and packet switched data calls.

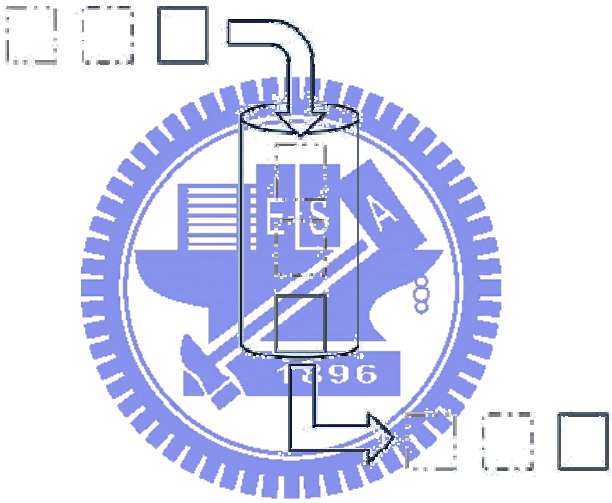


Figure 8. First In First Out Queueing

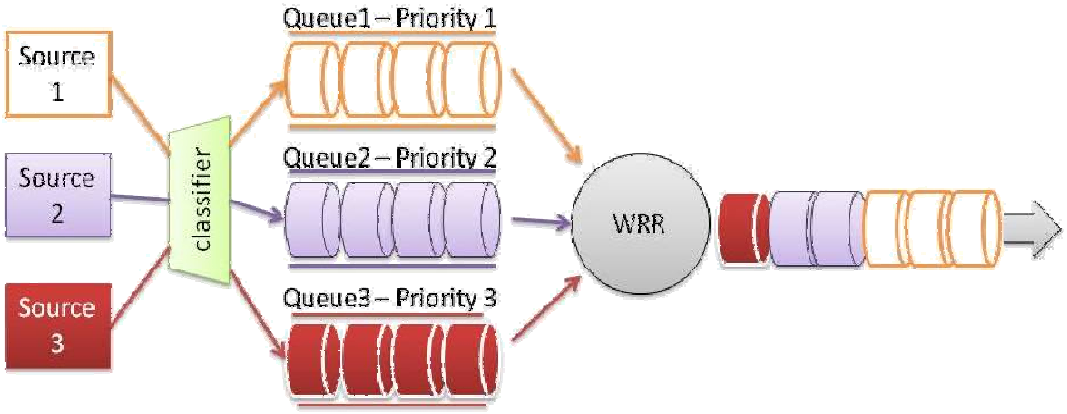


Figure 9. Weighted Round Robin Queueing

Chapter 4. An Approach to Priority-based Queueing Scheme with Two Queueing Buffer Allocations

It is impossible that a UMTS core network bandwidth can fully satisfy bandwidth requirements of all UMTS traffic. According to the QoS features of conversational, streaming, interactive, and background UMTS traffic and the packet transmission priority of each type of UMTS traffic, a priority-based queuing scheme with different queuing buffer allocations is proposed in this study to support a differentiated packet forwarding process for UMTS applications. The following subsections describe the processes of the enqueuing and dequeuing modules in details about the priority-based queuing scheme.

4.1. Queueing Buffer Allocations in Priority-based Queueing Scheme

Enqueuing and dequeuing modules are two major modules to handle packets enqueuing and dequeuing jobs in a queuing scheme. Usually, queuing buffer allocation in a queuing scheme affects packet forwarding processes in enqueuing and dequeuing modules directly [40, 45]. Considering packet forwarding in a DiffServ mechanism and packet forwarding starvation avoidance for four types of UMTS traffic, in [41] proposed assignment buffer access scheme and we use assignment buffer several queuing buffer allocation ideas, such as logical queuing buffer, guaranteed queuing buffer space, queuing buffer space dynamical allocation, and overflow queuing buffer space, to handle arrival packet enqueuing processes. With these queuing buffer allocation ideas, the proposed priority-based queuing scheme uses two queuing buffer allocations; one queuing buffer allocation is dynamic queuing buffer (DQB) allocation, the other is overflow queuing buffer (OQB) allocation. These two queuing buffer allocations will be described in details as the follows.

4.1.1. A Dynamic Queueing Buffer (DQB) Allocation

In the DQB allocation, a queuing buffer is divided into four logical queuing buffers, conversational, streaming, interactive, and background. Each logical queue buffer is FIFO queue; it stores its corresponding UMTS packets. Each logical queue buffer is divided into two segments: guaranteed buffer and dynamic buffer. For one type of UMTS packets, a guaranteed buffer stores packets unconditionally if space is available. Since each type of UMTS traffic has its own guaranteed buffer to enqueue arrival packets; it might reduce a possibility of a UMTS packet enqueuing starvation, especially for UMTS applications with lower packet transmission priorities. Moreover, a guaranteed buffer size depends on a packet transmission priority of its corresponding UMTS traffic. Usually, a logical queuing with a high packet transmission priority, more guaranteed buffer space would be allocated; this would be helpful for UMTS packets with higher packet transmission priorities to be enqueued easily.

When no space is available in a guaranteed buffer, a shared buffer will be allocated conditionally to store an arrival packet [46, 47]. A RED-based packet enqueuing process is invoked by the proposed enqueuing module. According to available space in queuing buffer and related parameters settings, the DQB allocation process is invoked to decide to enqueue or drop arrival packets. As an arrival packet is allowed to store in queuing buffer; it will be enqueued into its corresponding dynamic buffer [48, 49]. A dynamic buffer is the buffer space will be appended to a logical queuing buffer dynamically when one arrival packet is enqueued into it. A size setting of a dynamic buffer of each logical queuing buffer depends on its corresponding packet transmission priority. A logical queuing buffer with a high packet transmission priority might have a larger dynamic buffer than a logical queuing buffer with a low packet transmission priority. Figure 10 shows the diagram of the queuing buffer allocation in the DQB allocation.

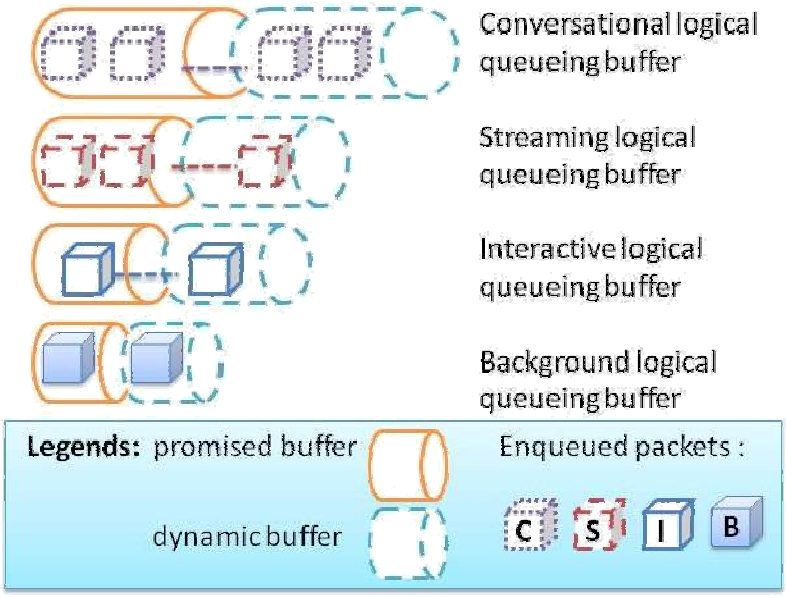


Figure 10. A diagram of queuing buffer with the DQB allocation

4.1.2. An Overflow Queueing Buffer (OQB) Allocation

Like the DQB allocation, a FIFO logical queuing buffer idea also is adopted in the OQB allocation. In the OQB allocation, five FIFO logical queuing buffers are used to enqueue arrival UMTS packets; four logical queuing buffers are guaranteed buffers and one logical queuing buffer is a shared buffer. The four guaranteed logical queuing buffers are corresponding to four types of UMTS traffic separately; an arrival packet will be enqueued into its corresponding guaranteed logical queuing buffer unconditionally when space is available to enqueue one packet. Allocations of these four guaranteed logical queuing buffers let four types of UMTS traffic can receive their queuing buffer space to enqueue their arrival packets; it can avoid packet enqueuing starvation.

Moreover, for enhancing an enqueuing possibility of UMTS packet with a higher packet transmission priority, these four logical queuing buffers depend on their packet transmission priorities to allocate their queuing buffer space. An overflow logical queuing buffer is one shared buffer; it is shared by all UMTS packets. A size of overflow logical queuing buffer is a difference between a physical queuing buffer size and

four guaranteed logical queuing buffers. For a UMTS arrival packet, if no space is available to accommodate one packet in its corresponding logical queuing buffer; the overflow queuing buffer will base on a RED-based packet enqueuing process to determine whether an arrival packet can be enqueued into the overflow queuing buffer or not. Usually, UMTS traffic with a higher packet transmission priority receives better settings in its corresponding RED-parameters to allow more packets with the same packet transmission priority to be enqueued into the overflow logical queuing buffer more easily. Figure 11 shows a diagram of the OQB allocation.

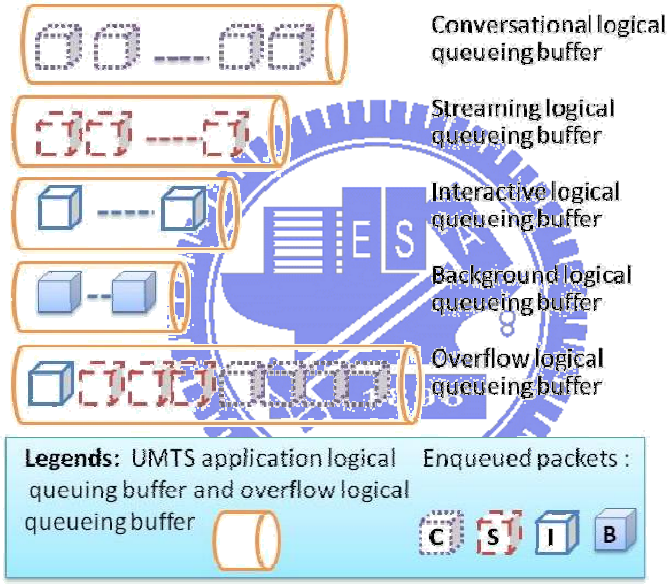


Figure 11. A diagram of queuing buffer with the OQB allocation

4.2. A Priority-Based Enqueueing Module

A priority-based packet enqueuing module [41, 50, 51] is adopted to handle arrival UMTS packets enqueuing process among several logical queuing buffers in the proposed queuing scheme. Since two types of queuing buffers are allocated to enqueue arrival packets; a two-stage packet enqueuing process is adopted to process an arrival packet enqueuing job in the priority-based enqueuing module. In the first stage packet enqueuing process, a FIFO method is used for a guaranteed buffer;

an arrival packet can be enqueued into a guaranteed buffer only if space is available in its corresponding guaranteed buffer. Otherwise, a RED-based packet enqueuing process is invoked in the second stage process to decide to enqueue or drop an arrival packet. Figure 12 shows the pseudo code of the two-stage packet enqueuing process.

A two stage priority-based enqueueing process

It bases on packet transmission priorities of four types of UMTS traffic to set the parameters: *guaranteed buffer sizes*, *minimum limits*, *maximum limits* and *packet enqueueing probabilities*, which are used in the priority-based enqueueing module

```
If    (queueing queue buffer space is available)
   { /* to invoke the first stage */
      if    (enqueued packets size < guaranteed buffer size)
         {
           To enqueue an arrival packet into a guaranteed buffer in its
           corresponding logical queue
      } else { /* to invoke the second stage */
               To invoke the RED-based packet enqueueing process
         }
} else { /* the queueing buffer is full */
         To drop an arrival packet
     }
```

Figure 12. A pseudo code of a two stage priority-based enqueuing process

The priority-based packet enqueuing module based on packet transmission priority of each type of UMTS traffic to set the parameters which are used in the two-stage packet enqueuing process. These parameters include guaranteed buffer size of each logical queue buffer and four groups of RED-alike parameters [33, 34]. Each group of RED-alike parameters consists of minimum limit, maximum limit, and packet enqueuing probability. Generally speaking, each logical queuing buffer depends on its packet transmission priority to receive corresponding guaranteed buffer size and RED-alike parameter settings. A logical queuing buffer with a higher packet transmission priority always receives more favorable parameter settings than logical queuing buffer with a lower packet transmission priority. It is easier for an arrival UMTS packet with a higher packet transmission priority to be enqueue

into its corresponding logical queuing buffer. By way of corresponding parameter settings, a DiffServ behavior can be supported by the proposed enqueuing module.

Since two queuing buffer allocations are adopted in the priority-based enqueuing module to handle an arrival packet enqueuing process, there exists some differences in the proposed packet enqueuing process. The proposed packet enqueuing process with the two queuing buffer allocations will be described in details as the follows.

4.2.1. A Priority-based Packet Enqueueing Module with a DQB Allocation

With the DQB allocation, two segments are allocated in each logical queuing buffer; one is a guaranteed buffer and the other is a dynamic buffer, it is a shared buffer. For enqueuing arrival packets into guaranteed buffer and dynamic buffer in a logical queuing buffer, the two-stage packet enqueuing process uses two different measures to process an arrival packet enqueuing job. In the first stage packet enqueuing process, a FIFO method is used for a guaranteed buffer; an arrival packet can be enqueued into a guaranteed buffer only if space is available in its corresponding guaranteed buffer. Otherwise, a RED-based packet enqueuing process is invoked in the second stage process to decide to enqueue or drop an arrival packet.

Packet enqueuing process in the second stage handles a dynamic buffer space allocation for an arrival packet. Since a dynamic buffer space allocation depends on available space of a physical queuing buffer; it is easier for an arrival packet to receive dynamic buffer space when more space is available in a physical queuing buffer. Thus, the RED-based packet enqueuing process bases on one group of RED-alike parameters which are corresponding to an arrival packet and available space in physical queuing buffer to determine whether dynamic buffer

space can be allocated or not. If dynamic buffer space can be allocated for an arrival packet; an arrival packet would be enqueued into its corresponding dynamic buffer. Allocated dynamic buffer space will be appended to the end of one logical queuing buffer which is corresponding to the arrival packet. Otherwise, an arrival packet will be dropped by the RED-based packet enqueuing process immediately.

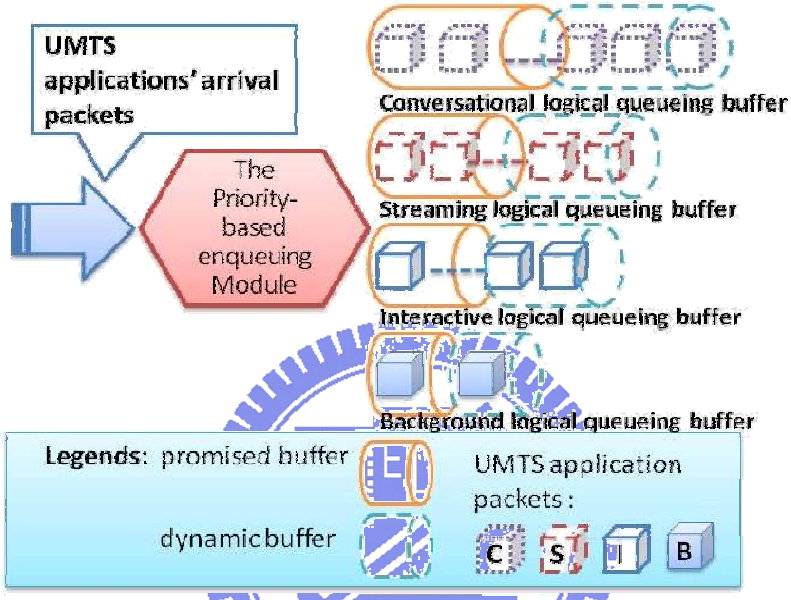


Figure 13. A diagram of arrival packets enqueueing process with the DQB allocation

In the RED-based packet enqueuing process, two thresholds [31, 33], minimum limit and maximum limit, about a physical queuing buffer utilization are use to decides whether an arrival packet can receive dynamic buffer space allocation unconditionally or to be dropped immediately. Dynamic buffer space can be allocated for an arrival packet unconditionally when length of enqueued packet is less than minimum limit. An arrival packet will be dropped immediately when length of enqueued packet is more than maximum limit. Moreover, Dynamic buffer space can be allocated for an arrival packet conditionally when length of enqueued packet is more than minimum limit and it is less than maximum limit. The diagram of arrival packets enqueueing process with the DQB allocation is shown in Figure 13 and Figure 14 shows the

pseudo code of the RED-based packet enqueuing process with the DQB allocation.

A RED-based packet enqueuing process with the DQB allocation

```
if (physical queuing buffer space is available)
   {
    if (the corresponding logical buffer size of an arrival packet  <
          its minimum limit)
     {
      To allocate dynamic buffer space with the DQB allocation and enqueue an
      arrival packet into the allocated dynamic buffer space
      To append to the allocated dynamic buffer space to one logical queuing
       buffer which is corresponding to the arrival packet
     } else {
              if (the corresponding logical buffer size of an arrival packet  ≧
                 its maximum limit )
                {
                 To drop an arrival packet immediately
              } else {
                       To generate a random probability based on a uniform
                         distribution
                         if (the random probability  ≦
                          the packet enqueuing probability )
                        {
                         To allocate dynamic buffer space with the DQB allocation
                           and enqueue an arrival packet into the allocated dynamic
                           buffer space
                          To append to the allocated dynamic buffer space to one
                           logical queuing buffer which is corresponding to the
                           arrival packet
                       } else {
                                To drop an arrival packet
                         }
                }
} else { /*no space available in queuing buffer */
        To drop an arrival packet immediately
 }
```

Figure 14. A pseudo code of the RED-based packet enqueuing process with the DQB allocation

4.2.2. A Priority-based Packet Enqueueing Module with an OQB Allocation

Like the priority-based enqueuing module with the DQB allocation, a two-stage packet enqueuing process is adopted in this enqueuing module, too. The process procedure of two-stage packet enqueuing process in this enqueuing module is same as the process procedure of two-stage packet enqueuing process with a DQB allocation. However, there exist differences in queuing buffer allocation between the DQB allocation and the OQB allocation; this causes that the two-stage packet enqueuing process has different process target with the different queuing buffer allocation.

Four logical queuing buffers which are corresponding to four types of UMTS traffic are guaranteed buffers; they are the guaranteed buffer space to store arrival UMTS packets with the OQB allocation. Guaranteed buffer space allocation of each type of UMTS traffic depends on its packet transmission priority. A UMTS application with higher packet transmission priority receives more guaranteed buffer space allocation. Differentiated guaranteed buffer space allocations exist among four types of UMTS traffic. FIFO method is used in the first stage process to handle arrival UMTS packets enqueuing job in guaranteed buffers. Arrival packets can be enqueued only if space is available in their corresponding guaranteed buffer; otherwise, a RED-based packet enqueuing process will be invoked in the second stage process to handle arrival packet enqueuing job in an overflow logical queuing buffer.

The second stage packet enqueuing process is a RED-based packet enqueuing process; it bases on the overflow queuing buffer's capacity and type of an arrival UMTS packet to handle arrival packet enqueuing job in the overflow logical queuing buffer. Like the RED-based packet enqueuing process with the DQB allocation, each type of UMTS traffic

has its corresponding RED-alike parameters minimum limit, maximum limit, and packet enqueuing probability, to calculate an enqueued probability of an arrival packet in this stage process. And, the settings of four groups of RED-alike parameters are corresponding to their packet transmission priorities. A packet with a higher packet transmission priority receives better settings in its corresponding RED-alike parameters and it can be enqueued into the overflow logical queuing buffer more easily. With proper parameters settings, a differentiated packet enqueuing behavior can be supported in the proposed enqueuing module with the OQB allocation. The diagram arrival packets enqueueing process with the OQB allocation is shown in Figure 15.

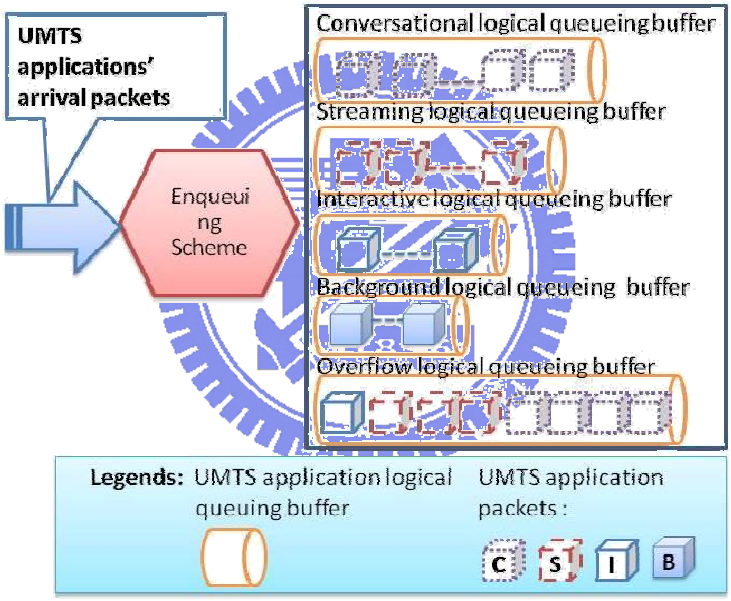


Figure 15. A diagram of arrival packets enqueueing process with the OQB allocation

Since there exists differences in a shared buffer allocation between the OQB allocation and the DQB allocation, the RED-based packet enqueuing process with the OQB allocation is a little different from to the RED-based packet enqueuing process with the DQB allocation. Essentially, these two RED-based packet enqueuing processes base on three parameters, minimum limit, maximum limit and packet enqueuing probability, to decide whether an arrival packet can be enqueued into a shared buffer or not. In the OQB allocation, an overflow logical queuing

buffer is the only shared buffer; it stores all enqueued packets which are allowed by the RED-based packet enqueuing process with the OQB allocation. A FIFO method is used to process packet enqueuing and dequeuing jobs in an overflow logical queuing buffer. A packet enqueuing sequence is determined by packet enqueuing time and four types of enqueued packets are mixed in an overflow logical queuing buffer. Figure 16 shows the pseudo code of the RED-based packet enqueuing process with the OQB allocation.

A RED-based packet enqueueing process with the OQB allocation

```
if (physical queueing buffer space is available)
 {
  if (enqueued packet size of the overflow logical buffer size  <
     a minimum limit of an arrival packet)
    {
     To enqueue an arrival packet into the overflow logical queueing buffer
     } else {
            if (enqueued packet size of the overflow logical buffer size  ≧
               its maximum limit of an arrival packet)
              {
               To drop an arrival packet immediately
            } else {
                    To generate a random probability based on a uniform distribution
                       if    (the random probability  ≦
                              the packet enqueueing probability )
                      {
                        To enqueue an arrival packet into the overflow logical
                          queueing buffer
                      } else {
                              To drop an arrival packet
}
                }
} else { /*no space available in queueing buffer */
        To drop an arrival packet immediately
  }
```

Figure 16. The pseudo code of a RED-based packet enqueueing process with the OQB allocation

4.3. A WRR Packet Dequeuing Module

A packet dequeuing module in a queuing scheme performs packets forwarding from its queuing buffer to packets' next hop gateways. Since all enqueued packets are stored in several logical queuing buffers no matter with the DQB allocation or the OQB allocation. Therefore, for supporting a DiffServ in packet deuqueuing process and avoiding packet dequeuing starvation, especially for packets with a lower transmission priority, a WRR idea is applied to propose a sequential-WRR (SWRR) dequeuing scheme in this packet dequeuing module [43]. A diagram of the SWRR dequeuing scheme with the DQB allocation is shown Figure 17 and a diagram of the SWRR dequeuing scheme with the OQB allocation is shown Figure 18.

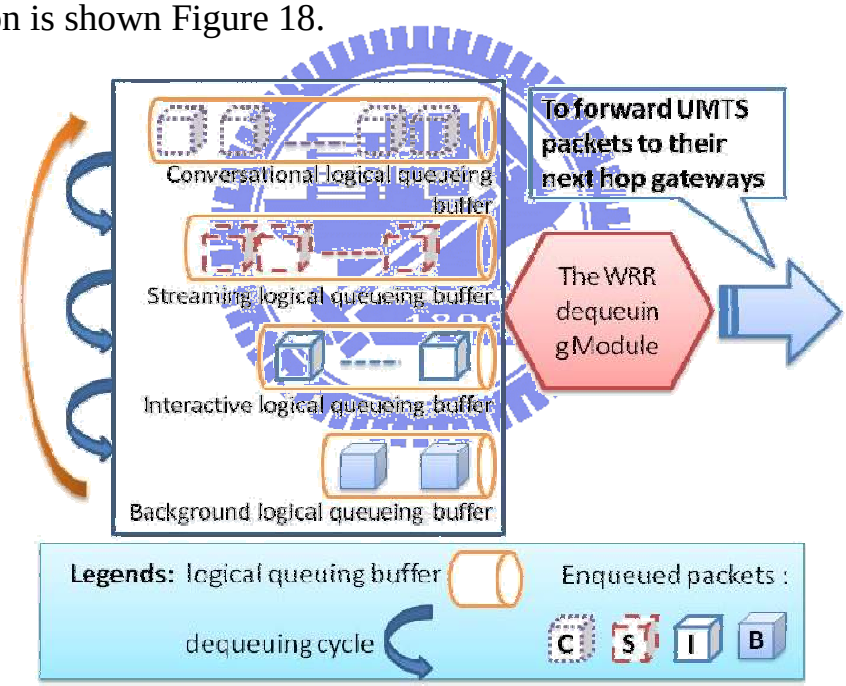


Figure 17. A diagram of the SWRR dequeuing module with the DQB allocation

In the SWRR scheme, each logical queuing buffer depends on its corresponding packet transmission priority to receive its deserved packet dequeuing weight in a weighted packet dequeuing round robin cycle. In a weighted packet dequeuing round robin cycle, each logical queuing depends on its deserved dequeuing weight to receive packet dequeuing turns in an assigned dequeuing sequence. One packet will be dequeued

from one logical queuing buffer when the logical queueing buffer receives a packet dequeuing turn. A dequeuing weight allotment bases on packet transmission priorities of logical queuing buffers and a number of logical queuing buffers to allot a corresponding packet dequeuing turns to each logical queuing buffer. Since the DQB allocation and the OQB allocation have different logical queuing buffers; a logical queuing buffer with the same packet transmission priority will receive different packet dequeuing turns in these two queuing buffer allocations. In the SWRR scheme, at least one packet, possibly more, will be dequeued from each logical queuing buffer in a packet dequeuing cycle. Usually, packets with a high transmission priority are more easily dequeued than packets with a low transmission priority. The packet dequeuing turns received by logical queuing buffers in the DQB allocation and the OQB allocation are shown in Table 3.

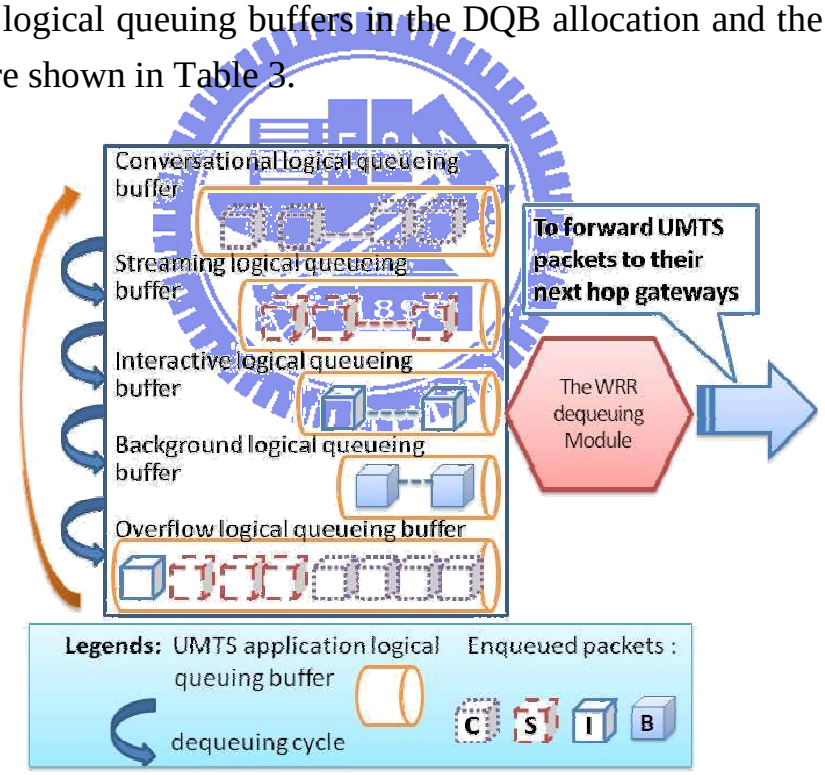


Figure 18. A diagram of the SWRR dequeuing module with the OQB allocation

Table 3. Packet dequeuing turns received by all logical queuing buffers with the DQB / OQB allocations in a packet dequeuing cycle

Logical queuing buffer	***Packet dequeuing turn number***	
	the DQB allocation	***the OQB allocation***
Conversational	4	5
Streaming	3	4
Interactive	2	2
Background	1	1
Overflow	0	3
Dequeueing cycle	10	15

The SWRR depends on a congruence equation to handle a dequeuing turn switching a process among all logical queuing buffers. Two variables, a packet dequeuing counter and a packet dequeuinging cycle, are the core parameters in the congruence equation. The packet dequeuing counter increments when one packet dequeued from one logical queuing buffer. The packet dequeuinging cycle is the sum of the assigned dequeuing turns received by all logical queuing buffers in a weighted packet dequeuing round robin cycle. Then, with a mapping relationship between a congruence which is calculated by the congruent equation and a packet dequeuing sequence among logical queuing buffers, dequeuing turns will be switched among all logical queuing buffer dispersedly and differential dequeuing turns can be received by logical queuing buffers.

$$a\ congruence\ = \\ packet\ dequeueing\ counter\ mod\ packet\ dequeueing\ cycle \qquad \cdots (2)$$

$$packet\ dequeueing\ cycle\ = \\ \sum_{i=1}^{total\ logical\ queueing\ buffer\ number} packet\ dequeueing\ turns\quad i \qquad \ldots.(3)$$

In additions, as one logical queueing buffer receives a packet dequeueing turn and there is no packet in the logical queueing buffer to be dequeued; a packet dequeueing turn transfer (PDTT) procedure will be invoked by the SWRR to accelerate packets dequeueing process. The

PDTT procedure will try to find one logical queueing buffer which has a higher packet transmission priority and has one packet can be dequeued. If one logical queueing buffer is found; it will receive a packet dequeueing turn which is transferred from another logical queueing buffer. With the PDTT procedure, one logical queueing buffer with a higher packet transmission priority always can receive a transferred packet dequeueing turn to dequeue one packet from itself. Thus, Figure 19 shows the flowchart of PDTT procedure which will be enhanced a differentiated packet dequeueing behavior. A flowchart of the proposed weighted round robin packet dequeueing module is shown in Figure 20.

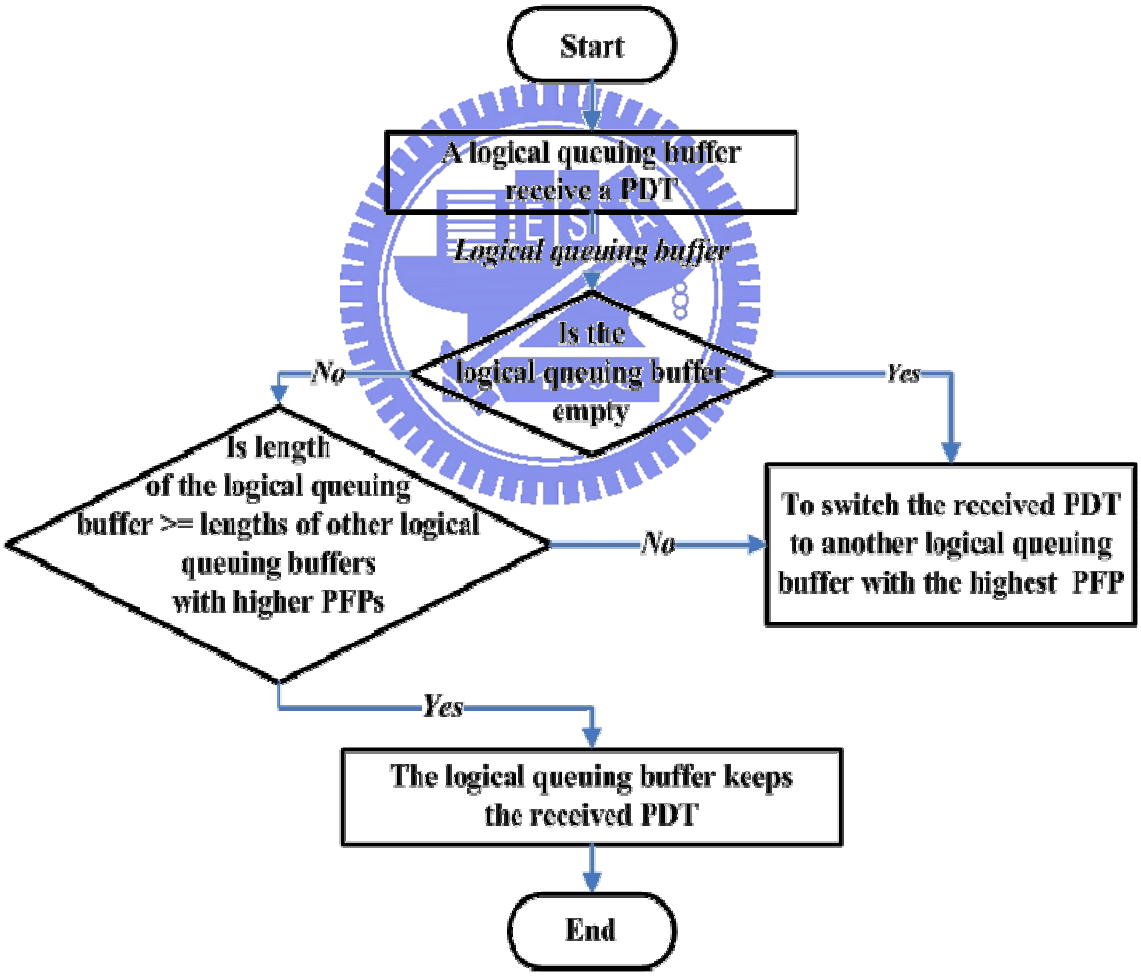


Figure 19. Flowchart of dequeueing turn transfer PDTT procedure

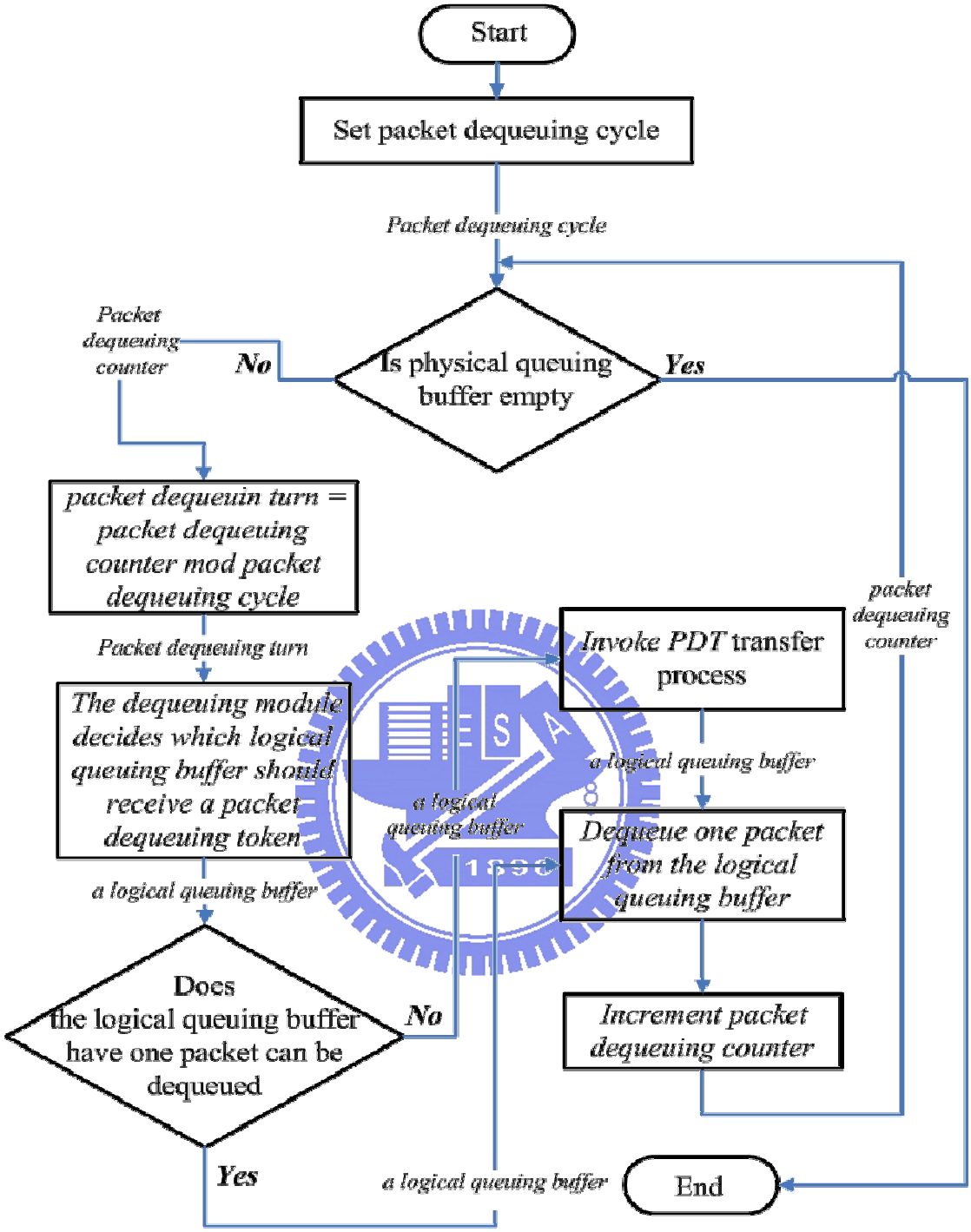


Figure 20. Flowchart of the WRR packet dequeueing module

Chapter 5. Simulation and Results Analysis

The Network Simulator – ns2 is a discrete event simulator targeted at network research [52]. Ns2 provides substantial support for simulation of TCP/UDP, routing, and multicast over wired and wireless (local, mobile and satellite) networks. In this study, the ns2 (version 2.26) is used as the simulation platform in this study.

The proposed queueing scheme with the DQB allocation or with the OQB allocation is implemented in the ns2 with C++; moreover, UMTS applications are coded with C++ and integrated in the ns2, too. Several simulation scenarios are coded with TCL and simulated to observe packet forwarding performance of the proposed queueing scheme with the DQB allocation or with the OQB allocation. Moreover, with the simulation results, packet forwarding performance with the DQB allocation or with the OQB allocation is compared and analyzed.

5.1. Simulation Topology and Parameters

For analyzing simulation data more concisely, a simplified topology is used to simulate a UMTS core network and simplified UMTS traffic is simulated over four pairs of UMTS connections; each connection represents one type of UMTS traffic. The simulation topology is shown in Figure 21. The backbone bandwidth over a UMTS core network is 2MB and the bandwidth requirement of each UMTS connection is 1MB.

Several dimensions of simulation parameters, four types of UMTS traffic, packet size, backbone bandwidth, bandwidth requirements, and traffic transmission patterns, are used to construct various simulation scenarios. Moreover, since the DQB allocation and the OQB allocation are a RED-based queueing buffer allocation; both of them require four groups of RED-alike parameters to handle four types of arrival UMTS packets in a packet enqueueing process. Due to a differentiated packet transmission behavior is required among four types of UMTS traffic;

therefore, the settings of these four groups of RED-alike parameters base on their corresponding packet transmission priorities. Finally, two traffic transmission patterns, continuous and intermittent, will be simulated. The settings of these parameters are listed in Table 4.

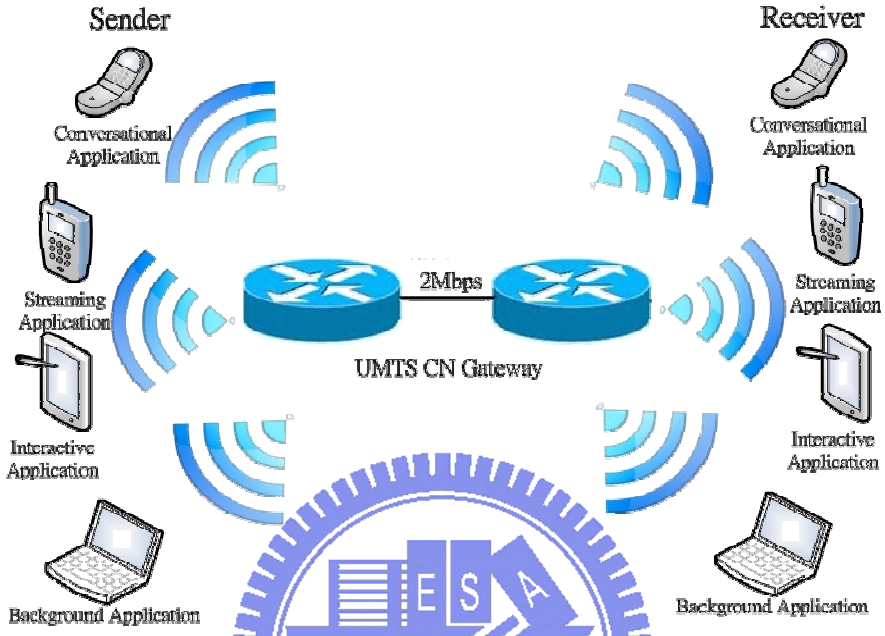


Figure 21. A diagram of the simulation topology

Table 4. A summary of simulation parameters

Parameter dimensions	***Setting values***				
Four types of UMTS traffic	**CON, STR, INT, BAC**				
Packet size	100 bytes, 500 bytes, 1000 bytes				
Two-phase queueing buffer allocation scheme settings		**GBS**	**MIN**	**MAX**	**PEP**
	CON	8	8	40	1.00
	STR	7	7	38	0.95
	INT	5	5	35	0.85
	BAC	2	2	31	0.70
UMTS traffic bandwidth requirement	1.0MB	***Backbone bandwidth in the UMTS core network***			2.0MB
Queueing buffer allocation	**DQB, OQB**				
Traffic transmission pattern	Continuous, Intermittent				
Simulation time	30 seconds				

Legends:
CON: Conversational traffic, **STR**: Streaming traffic, **INT**: Interactive traffic,
BAC: Background traffic, **GBS**: Guaranteed buffer size,
PEP: packet enqueueing probability, **MIN**: minimum limit, **MAX**: maximum limit

5.2. Simulation Results and Analysis

According to the traffic transmission patterns, simulation scenarios can be divided into two parts, continuous and intermittent. Moreover, for having a packet forwarding performance comparison between the DQB allocation and the OQB allocation, the others parameters settings are the same in these two parts simulations and simulation results in the same traffic transmission pattern are collected, compared, and analyzed. The simulation results are described in the following subsections.

5.2.1. UMTS Packet Transmission in a Continuous Traffic Pattern

In this section, all scenarios bases on the parameters settings shown in Table 4 and all UMTS packets are transmitted continuously. These scenarios intend to observe the UMTS packet forwarding performance when the backbone bandwidth is insufficient and four types of UMTS traffic keep transmitting continuously. The simulation results about UMTS packet enqueued and dequeued performance information are shown in Table 5 and Figure 22.

By examining Table 5 and Figure 22, it is obvious that a differentiated packet forwarding behavior is supported by the proposed queueing scheme with the DQB / OQB allocations; each type of UMTS traffic receives its packet forwarding performance corresponding to its packet transmission priority. Conversational and OQB allocations; each type of UMTS traffic receives its packet forwarding performance corresponding to its packet transmission priority. Conversational and streaming applications receive better packet forwarding performance than interactive and background applications. Comparing packet dequeueing volumes reached by four types of UMTS traffic with the two allocations, we can find that conversational traffic receives the same packet dequeueing volumes with both the DQB and OQB allocations; streaming traffic receives approximately the same packet dequeueing

volumes with both the DQB and OQB allocations; interactive traffic receives better dequeueing volumes with the DQB allocation; background traffic receives better dequeueing volumes with the OQB allocation. Moreover, by examining the simulation trace files in details, there is packet starvation for background traffic with the DQB allocation; only a few background packets can be forwarded at the beginning of packet dequeueing process and most of background packets are dropped during a packet dequeueing process.

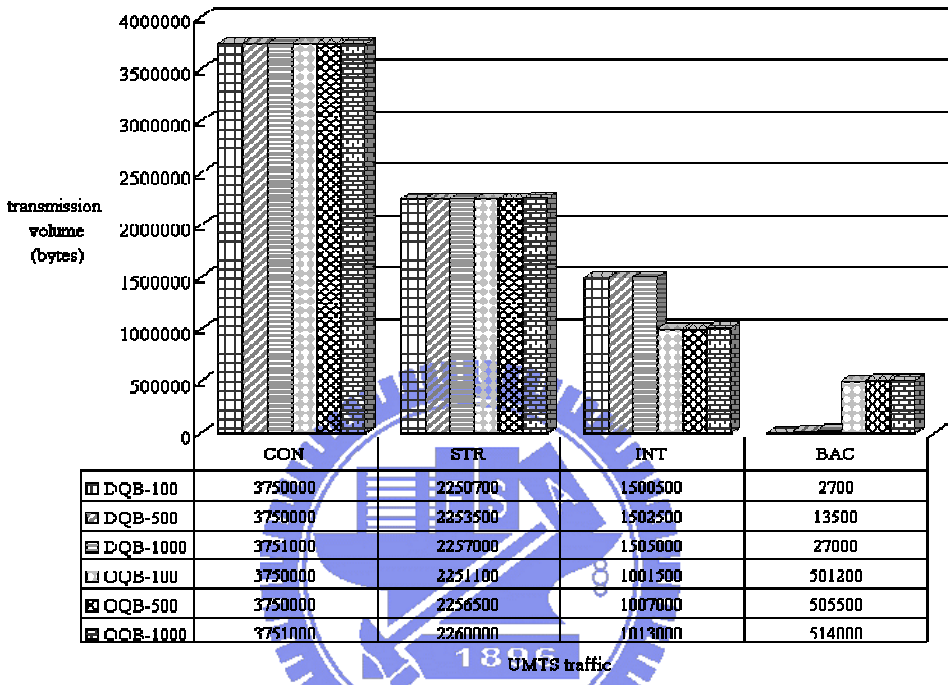
Table 5. A packet enqueueing / dequeueing statistic of UMTS traffic in a continuous transmission pattern

UT	**PS**	***Arrival packets***		***Enqueued packets***		***Dequeued packets***		***Dropped packets***		***Packet dequeued ratio (%)***		***Packet dropped ratio (%)***	
		DQB	**OQB**	**DQB**	**OQB**	**DQB**	**OQB**	**DQB**	**OQB**	**DQB**	**OQB**	**DQB**	**OQB**
CON	**100**	37500	37500	37500	37500	37500	37500	0	0	100	100	0	0
	500	7500	7500	7500	7500	7500	7500	0	0	100	100	0	0
	1000	3751	3751	3751	3751	3751	3751	0	0	100	100	0	0
STR	**100**	37500	37500	22507	22511	22507	22511	14993	14989	60.02	60.03	39.98	39.97
	500	7500	7500	4507	4513	4507	4513	2993	2987	60.09	60.17	39.91	39.83
	1000	3751	3751	2257	2260	2257	2260	1494	1491	60.17	60.25	39.83	39.75
INT	**100**	37500	37500	15005	10015	15005	10015	22495	27485	40.01	26.71	59.99	73.29
	500	7500	7500	3005	2014	3005	2014	4495	5486	40.07	26.85	59.93	73.15
	1000	3751	3751	1505	1013	1505	1013	2246	2738	40.12	27.01	59.88	72.99
BAC	**100**	37500	37500	27	5012	27	5012	37473	32488	0.072	13.37	99.93	86.63
	500	7500	7500	27	1011	27	1011	7473	6489	0.36	13.48	99.64	86.52
	1000	3751	3751	27	514	27	514	3724	3237	0.72	13.70	99.28	86.30

Legends:
UT: UMTS traffic, **PS**: packet size,
CON: conversational, **STR**: streaming, **INT**: interactive, **BAC**: background

Examining Figure 22, it can be found that there exists a little difference in packet dequeueing volume with these two allocations when UMTS packets are transmitted in different size. Four types of UMTS traffic can receive better packet dequeueing volume separately when UMTS traffic is transmitted in a larger packet size, especially for UMTS

traffic with lower packet transmission priorities. Further comparing total packet dequeueing throughputs supported by these two allocations, only a very small difference exists in the total packet dequeueing throughputs. The DQB allocation outperforms the OQB allocation; it becomes more obvious when UMTS traffic is transmitted in a large packet size.



Legends:
CON: conversational, **STR**: streaming, **INT**: interactive, **BAC**: background,
DQB-100: packet size = 100 bytes and allocation = **DQB**,
DQB-500: packet size = 500 bytes and allocation = **DQB**,
DQB-1000: packet size = 1000 bytes and allocation = **DQB**,
OQB-100: packet size = 100 bytes and allocation = **OQB**,
OQB-500: packet size = 500 bytes and allocation = **OQB**,
OQB-1000: packet size = 1000 bytes and allocation = **OQB**

Figure 22. A statistics of packet dequeueing volume in a continuous transmission pattern

Table 6. An average packet delay statistic of UMTS traffic in a continuous transmission pattern

UMTS Traffic		***Conversational***		***Streaming***		***Interactive***		***Background***	
Allocations		**DQB**	**OQB**	**DQB**	**OQB**	**DQB**	**OQB**	**DQB**	**OQB**
PS (B)	**100**	0.8006	0.8005	1.3333	1.3334	2.0	2.9968	4.0	5.9890
	500	4.0147	4.0112	6.6667	6.6640	10.0	14.9285	76.9231	29.7564
	1000	8.0587	8.0437	13.3333	13.3351	20.0	29.7470	153.8462	58.6901

Legends:
Unit : *ms*, **PS** : packet size, **B** : bytes

Table 7. An average packet jitter statistic of UMTS traffic in a continuous transmission pattern

UMTS Traffic		***Conversational***		***Streaming***		***Interactive***		***Background***	
Allocation		**DQB**	**OQB**	**DQB**	**OQB**	**DQB**	**OQB**	**DQB**	**OQB**
PS (B)	100	0.5654	0.117	266.6607	364.6541	800.0	401.2484	0.0	5.0299
	500	14.1371	2.934	1333.1853	1829.306	4000.0	2035.785	59200.0	132.8048
	1000	57.22	11.7365	2666.0754	3638.6182	8000.0	4185.9545	118400.0	648.43775

Legends:
Unit : μs, **PS** : packet size, **B** : bytes

Table 6 and Table 7 show average packet delays and average packet jitters received by four types of UMTS traffic with the two buffer allocations in a continuous transmission pattern. Examining Table 6, we can find that average packet delays received by UMTS traffic are corresponding to their packet transmission priority; conversational and streaming traffic receive better average packet delays than interactive and background traffic. Looking over average packet delays received by four types of UMTS traffic, conversational and streaming traffic receive close average packet delays with both the DQB and OQB allocations; interactive traffic receives better average packet delays with the DQB allocation; background traffic receives better average packet delays with the OQB allocation when transmitted packets are larger. Viewing average packet jitters received by four types of UMTS traffic, conversational and interactive traffic receive better average packet jitters with the OQB allocation; streaming traffic receives better average packet

jitters with the DQB allocation; and, background traffic receives better average packet jitters with the OQB allocation when transmitted packets are larger. Moreover, we also find that average packet jitter of background traffic is 0 μs with the DQB allocation when transmitted packet size is 100 bytes; it seems an unusual simulation result. Further examining the simulation trace data that all UMTS packets are transmitted in 100 bytes , we can find that background UMTS packets can be transmitted stably before a background packet forwarding starvation occurs (i.e., bandwidth is available for background packets to forward to their next hop gateway). The background packet forwarding starvation should be the reason why the average packet jitter of background traffic is 0 μs; it is a special case about average packet jitters of UMTS traffic.

After examining Table 6 and Table 7, except for background traffic, an average packet delay / jitter received by one type of UMTS traffic depends on its corresponding packet size; UMTS traffic receives a smaller average packet delay / jitter when transmitted packet is smaller; on the contrary, UMTS traffic receives a larger average packet delay / jitter when transmitted packet is larger. For most of UMTS traffic, there exists a ratio relationship between average packet delay / jitter and packet size with the DQB / OQB allocations; this ratio relationship can be expressed as the following equations.

$$A\ ratio_{ut,\ packet\ delay} \approx \frac{packet\ size_{ut,\ large}}{packet\ size_{ut,\ small}} \approx \frac{an\ average\ packet\ delay_{ut,large}}{an\ average\ packet\ delay_{ut,small}} \quad \text{.....(4)}$$

$$A\ ratio_{ut,\ packet\ jitter} \approx \frac{packet\ size_{ut,\ large}}{packet\ size_{ut,\ small}} \approx \frac{an\ average\ packet\ delay_{ut,large}}{an\ average\ packet\ delay_{ut,small}} \quad \text{.....(5)}$$

where ut = conversational traffic, streaming traffic, and interactive traffic

Moreover, after looking over the simulation trace files in details, we find that there exists a disorder packet transmission issue for UMTS traffic with the OQB allocation; there is no disorder packet transmission for UMTS traffic with the DQB allocation. The order of severity about the disorder packet transmission among UMTS traffic depend on the packet transmission priority of each type of UMTS traffic; UMTS traffic with a higher packet transmission priority might transmit its packets more disorderly than UMTS traffic with a lower packet transmission priority. A statistic of disorder packet transmission among UMTS traffic with the OQB allocation in a continuous traffic pattern is shown in Table 8.

Table 8. A statistic of disorder packet transmission among UMTS traffic with the OQB allocation in a continuous traffic pattern

UMTS Application	***Conversational***			***Streaming***			***Interactive***			***Background***		
Packet size	**100**	**500**	**1000**	**100**	**500**	**1000**	**100**	**500**	**1000**	**100**	**500**	**1000**
Number of DPT	12484	2484	1234	2503	506	253	8	6	6	5	5	6
Percentage of DPT (%)	33.29	33.12	32.91	6.68	6.75	6.75	0.02	0.08	0.16	0.013	0.07	0.16

Legends:
DPT: disorder packet transmission

At last, a simple packet forwarding performance weight measure is adopted to evaluate four types of UMTS traffic's packet forwarding performance with the two allocations. Four evaluation items, dequeued packets, dropped packets, average packet delays, and average packet jitters which are corresponding to QoS features, are used in the weight measure. Each type of UMTS traffic depends on its packet forwarding performance ranking in each evaluation item to receive its corresponding weight in the weight measure. A total packet forwarding performance ranking assignment depends on the sum of weights in the four evaluation items; one type of UMTS traffic will receive a better total packet forwarding performance ranking only when it receive a larger weight

sum. Since three kinds of packet sizes are used to understand packet forwarding performance of four types of UMTS traffic in the scenarios; the weight measure also bases on the packet sizes to evaluate the packet forwarding performance received by UMTS traffic. A statistic of packet forwarding performance weights and rankings about four types of UMTS traffic in a continuous traffic pattern is listed in Table 9.

Table 9. A ranking weight statistic of UMTS application with both of the DQB allocation and the OQB allocation in a continuous traffic pattern

Allocation		***DQB***				***OQB***			
PS	***EI***	***CON***	***STR***	***INT***	***BAC***	***CON***	***STR***	***INT***	***BAC***
100 bytes	***DP***	4	3	2	1	4	3	2	1
	DRP	4	3	2	1	4	3	2	1
	APD	4	3	2	1	4	3	2	1
	APJ	3	2	1	4	4	2	1	3
	WS	15	11	7	7	15	11	7	6
	TPFPR	**1**	**2**	**3**	**3**	**1**	**2**	**3**	**4**
500 bytes	***DP***	4	3	2	1	4	3	2	1
	DRP	4	3	2	1	4	3	2	1
	APD	4	3	2	1	4	3	2	1
	APJ	4	3	2	1	4	2	1	3
	WS	16	12	8	4	16	11	7	6
	TPFPR	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
1000 bytes	***DP***	4	3	2	1	4	3	2	1
	DRP	4	3	2	1	4	3	2	1
	APD	4	3	2	1	4	3	2	1
	APJ	4	3	2	1	4	2	1	3
	WS	16	12	8	4	16	11	7	6
	TPFPR	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**

Legends:
CON: conversational, **STR**: streaming. **INT**: interactive, **BAC**: background,
PS: packet size, **EI**: evaluation item, **DP**: dequeued packets, **DRP**: dropped packets,
WS: weight sum, **APD**: average packet delay, **APJ**: average packet jitter,
TPFPR: total packet forwarding performance ranking

Examining Table 9, we can find that four types of UMTS traffic depend on their packet transmission priorities to receive their deserved packet forwarding performance with the DQB and OQB allocations; no matter what kind of packet sizes are transmitted. A differentiated packet

forwarding behavior can be supported by the proposed queuing scheme with the two allocations in a continuous transmission traffic pattern.

5.2.2. UMTS Packet Transmission in an Intermittent Traffic Pattern

Actually, it is impossible for all UMTS traffic to be transmitted in a continuous traffic pattern all the time. Four types of UMTS traffic would be transmitted intermittently and it presents an intermittent traffic pattern during one specific period. This scenario tries to observe forwarding performance among UMTS traffic supported by the proposed queuing scheme with both the DQB allocation and the OQB allocation when the bandwidth requirement of all UMTS traffic exceeds the backbone bandwidth over a UMTS core network and four types of UMTS traffic is transmitted intermittently. The parameter settings in this scenario also bases on Table 4. The statistics of enqueued packets and dequeued packets are listed in Table 10 and rates of UMTS packets dequeuing are shown in Figure 23.

Examining Table 10 and Figure 23, we can find that four types of UMTS traffic depend on their packet transmission priorities to receive their deserved dequeuing packet rates with the DQB and OQB allocations; differentiated packet forwarding rates can be supported by the proposed queuing scheme with the two allocations in an intermittent transmission pattern. Comparing packet dequeuing rates of four types of UMTS traffic, conversational traffic, streaming, and background traffic receive better performance with the OQB allocation; interactive traffic receives better performance with the DQB allocation. Moreover, we also find that packet forwarding starvation of background traffic in a continuous transmission pattern can be released in an intermittent transmission pattern with both the DQB allocation and the OQB allocation.

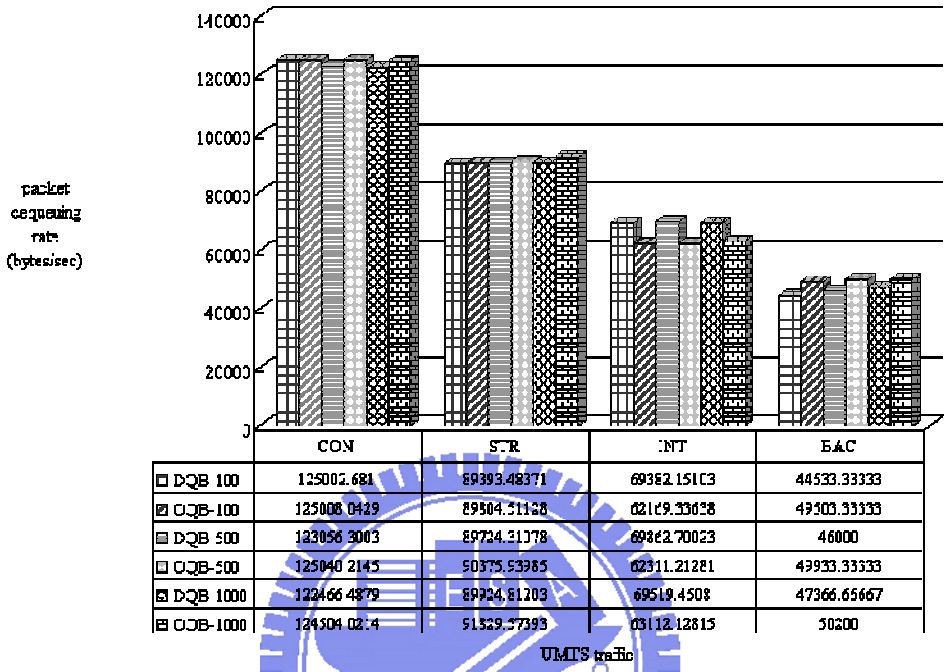
Table 10. A packet enqueuing / dequeuing statistic of UMTS traffic in an intermittent transmission pattern

UT	*PS*	*Arrival packets*		*Enqueued packets*		*Dequeued packets*		*Dropped packets*		*Packet dequeued ratio (%)*		*Packet dropped ratio (%)*		*TT (second)*
		DQB	**OQB**	**DQB**	**OQB**	**DQB**	**OQB**	**DQB**	**OQB**	**DQB**	**OQB**	**DQB**	**OQB**	
C	**100**	23314	23314	23313	23314	23313	23314	1	0	100	100	0	0	
O	**500**	4664	4664	4590	4664	4590	4664	74	0	98.41	100	1.59	0	18.65
N	**1000**	2334	2334	2284	2322	2284	2322	50	12	97.86	99.49	2.14	0.51	
S	**100**	24939	24939	17834	17916	17834	17916	7105	7023	71.51	71.84	28.49	28.16	
T	**500**	4989	4989	3580	3606	3580	3606	1409	1383	71.76	72.28	28.24	27.72	19.95
R	**1000**	2497	2497	1794	1832	1794	1832	703	665	71.85	73.37	28.15	26.63	
I	**100**	27314	27314	15160	13584	15160	13584	12154	13730	55.50	49.73	44.50	50.27	
N	**500**	5464	5464	3053	2723	3053	2723	2411	2741	55.87	49.84	44.13	50.16	21.85
T	**1000**	2733	2733	1519	1379	1519	1379	1214	1354	55.58	50.46	44.42	49.54	
B	**100**	37500	37500	13360	14851	13360	14851	24140	22649	35.63	39.60	64.37	60.40	
A	**500**	7500	7500	2760	2996	2760	2996	4740	4504	36.80	39.95	63.20	60.05	30.0
C	**1000**	3751	3751	1421	1506	1421	1506	2330	2245	37.88	40.15	62.12	59.85	

Legends:
UT: UMTS traffic, **PS**: packet size, **TT**: transmission time
CON: conversational, **STR**: streaming. **INT**: interactive, **BAC**: background

Table 11 and Table 12 show average packet delays and average packet jitters received by four types of UMTS traffic with the two dynamic buffer allocations in an intermittent transmission pattern. Looking over Table 11, we can find that average packet delays received by four types of UMTS traffic are corresponding to their packet transmission priorities with the two dynamic buffer allocations. Moreover, we also find that the conversational and streaming traffic receive better average packet delay with the OQB allocation and the interactive traffic receives better average packet delay with the DQB allocation in an intermittent transmission pattern. Examining Table12, we can find that the average packet jitters received by four types of UMTS traffic seem not corresponding to their packet transmission priorities with both the DQB and OQB allocations in an intermittent transmission pattern; the conversational traffic receives the best average packet jitters and the streaming traffic receives better average packet

jitters than the interactive traffic. And, except for the streaming traffic, the conversational, interactive, and background traffic can receive better average packet jitters with the OQB allocation in an intermittent packet transmission patter.



Legends:

CON: conversational, STR: streaming, **INT**: interactive, **BAC**: background,
DQB-100: packet size = 100 bytes and allocation = **DQB,**
DQB-500: packet size = 500 bytes and allocation = **DQB**,
DQB-1000: packet size = 1000 bytes and allocation = **DQB**,
OQB-100: packet size = 100 bytes and allocation = **OQB**,
OQB-500: packet size = 500 bytes and allocation = **OQB**,
OQB-1000: packet size = 1000 bytes and allocation = **OQB**

Figure 23. A statistics of packet dequeuing rate in an intermittent transmission pattern

Examining Table 11 and Table 12, we find that there exists a ratio relationship between average packet delay and packet size with both the DQB and OQB allocations in an intermittent transmission pattern; this ratio relationship is similar to equation 4; we also find that UMTS traffic with a smaller packet size can receive better average packet jitter.

Table 11. An average packet delay statistic of UMTS traffic in an intermittent transmission pattern

UMTS Traffic		*Conversational*		*Streaming*		*Interactive*		*Background*	
allocations		**DQB**	**OQB**	**DQB**	**OQB**	**DQB**	**OQB**	**DQB**	**OQB**
PS (B)	**100**	0.8045	0.8045	1.1212	1.1171	1.4464	1.6182	2.0290	2.0207
	500	4.1758	4.1116	5.6320	5.6263	7.2857	8.2523	9.9790	10.0254
	1000	8.6228	8.4625	11.3721	11.2713	14.9069	16.6284	21.1606	19.9867

Legends:
Unit : *ms*, **PS** : packet size, **B** : bytes

Table 12. An average packet jitter statistic of UMTS traffic in an intermittent transmission pattern

UMTS traffic		*conversational*		*streaming*		*interactive*		*background*	
allocations		**DQB**	**OQB**	**DQB**	**OQB**	**DQB**	**OQB**	**DQB**	**OQB**
PS (B)	**100**	4.4626	2.3513	166.3637	224.9944	496.8321	251.002	1390.7471	63.2231
	500	175.0327	51.9759	847.7044	1138.4102	2536.6174	1523.3886	4020.3046	715.4309
	1000	627.4165	247.1910	1820.8287	2333.3333	5310.3905	3422.3195	5085.2713	2114.3617

Legends:
Unit : μs, **PS** : packet size, **B** : bytes

Table 13. A statistic of disorder packet transmission among UMTS traffic with the OQB allocation in an intermittent traffic pattern

UMTS Application	*Conversational*			*Streaming*			*Interactive*			*Background*		
Packet size	**100**	**500**	**1000**	**100**	**500**	**1000**	**100**	**500**	**1000**	**100**	**500**	**1000**
Number of DPT	5768	1114	521	1564	319	169	2018	424	214	364	198	161
Percentage of DPT (%)	24.74	23.9	22.33	6.272	6.4	6.78	7.39	7.76	7.83	0.971	2.64	4.3

Legends:
DPT: disorder packet transmission

In additions, the disorder packet transmission issue for UMTS traffic still exists with the OQB allocation in an intermittent transmission pattern. A statistic of disorder packet transmission among UMTS traffic with the OQB allocation in an intermittent traffic pattern is shown in Table 13 .The disorder packet transmission of the conversational traffic is the most serious and the disorder packet transmission of the

background traffic is the least serious. With the OQB allocation, packet transmission priority might impact order of severity about the disorder packet transmission among UMTS traffic in an intermittent transmission pattern.

Table 14. A ranking weight statistic of UMTS application with both of the DQB/OQB allocation in an intermittent traffic pattern

Allocation		***DQB***				***OQB***			
PS	***EI***	***CON***	***STR***	***INT***	***BAC***	***CON***	***STR***	***INT***	***BAC***
100 bytes	***DPR***	4	3	2	1	4	3	2	1
	DRPR	4	3	2	1	4	3	2	1
	APD	4	3	2	1	4	3	2	1
	APJ	4	3	2	1	4	2	1	3
	WS	16	12	8	4	16	11	7	6
	TPFPR	**1**	**2**	**3**	**3**	**1**	**2**	**3**	**4**
500 bytes	***DPR***	4	3	2	1	4	3	2	1
	DRPR	4	3	2	1	4	3	2	1
	APD	4	3	2	1	4	3	2	1
	APJ	4	3	2	1	4	2	1	3
	WS	16	12	8	4	16	11	7	6
	TPFPR	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
1000 bytes	***DPR***	4	3	2	1	4	3	2	1
	DRPR	4	3	2	1	4	3	2	1
	APD	4	3	2	1	4	3	2	1
	APJ	4	3	1	2	4	2	1	3
	WS	16	12	7	5	16	11	7	6
	TPFPR	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**

Legends:
CON: conversational, **STR**: streaming. **INT**: interactive, **BAC**: background, **PS**: packet size, **EI**: evaluation item, **DPR**: dequeued packet ratio, **DRPR**: dropped packet ratio, **APD**: average packet delay, **WS**: weight sum, **APJ**: average packet jitter, **TPFPR**: total packet forwarding performance ranking

Finally, the packet forwarding performance weight measure is used evaluate four types of UMTS traffic's packet forwarding performance with the two allocations in an intermittent packet transmission pattern. A statistic of packet forwarding performance weights and rankings about

four types of UMTS traffic in an intermittent traffic pattern is listed in Table 14.

Examining the total packet forwarding performance ranking shown in Table 14, we can find that four types of UMTS traffic can receive their deserved packet forwarding performance with the DQB and OQB allocations; no matter what kind of packet sizes are transmitted. A differentiated packet forwarding behavior can be supported by the proposed queuing scheme with the two allocations in an intermittent transmission traffic pattern.

5.2.3. Summary

After examining the simulation results, several summaries about the proposed queuing scheme with the two buffer allocations are received and described as the fellows.

- The proposed queuing scheme with both the DQB and OQB allocations can support a differentiated packet forwarding behavior for four types of UMTS traffic either in a continuous transmission pattern or an intermittent transmission pattern; four types of UMTS traffic can depend on their packet transmission priorities to receive their deserved packet forwarding performance.
- Comparing the packet forwarding performance received by four types of UMTS traffic with both the DQB and OQB allocations, we can find that the dequeued / dropped packet volume of UMTS packets is almost the same in a continuous transmission pattern and the dequeued / dropped packet volume of UMTS traffic is very close in an intermittent transmission pattern. We also find that the conversational and streaming traffic receives close average packet delays with both the DQB and OQB allocations and it is easy for UMTS traffic to receive better average packet jitters with the OQB allocation either in a continuous transmission pattern or in an

intermittent transmission pattern.

- After examining the packet forwarding performance of four types of UMTS traffic and packet size, we find that there exists a close relation among them. UMTS traffic receives more dequeuing packet volume when transmitted packet size is large; however, UMTS traffic receives larger average packet delays and jitters. On the contrary, UMTS traffic receives less dequeuing packet volume when transmitted packet size is small; but, UMTS traffic receives smaller average packet delays and jitters.
- Finally, although the proposed queuing scheme with both the DQB and OQB allocations can forward four types of UMTS packets in a DiffServ way; but, for either the DQB allocation or the OQB allocation, some issues need be watched. For the DQB allocation, a packet forwarding starvation received by the background application in the continuous traffic pattern; but, the packet forwarding starvation among UMTS traffic with lower packet transmission priorities might be released when UMTS packets are transmitted in an intermittent transmission pattern. For the OQB allocation, disorder packet transmissions occur among UMTS traffic both in a continuous transmission pattern and an intermittent transmission pattern. The overflow logical queuing buffer operation in the SWRR scheme should be the reason why UMTS packets are transmitted disorderly. In spite of disorder packet transmissions in an IP layer can be recovered in a transport layer; the disorder packet transmission still is an issue for the proposed queuing scheme to refine the OQB allocation.

Chapter 6. Conclusions

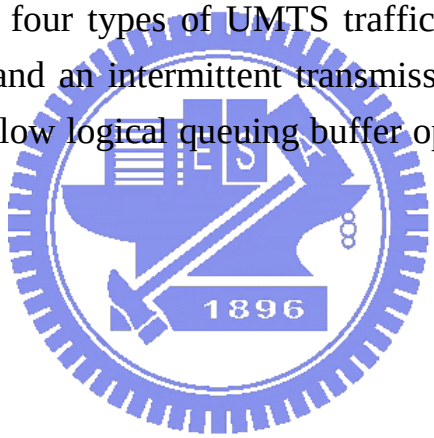
As a UMTS system enters an all-IP stage, a packet switching scheme is used to forward packets from all UMTS applications within a UMTS core network. Since four types of UMTS traffic defined by the 3GPP are supported by the UMTS; each type of UMTS traffic has its QoS features and requirements. However, bandwidth resource of a UMTS core network is limited; it might be insufficient for UMTS applications to receive their required QoS. For satisfying most of UMTS applications' QoS requirements, a queuing scheme should be adopted to handle UMTS packet forwarding process in a DiffServ way within a gateway over a UMTS core network.

This study proposes a priority-based queuing scheme to support differentiated packet forwarding behaviors among UMTS traffic within a UMTS core network. In the proposed queuing scheme, two queuing buffer allocations, the DQB and OQB, are adopted to handle packet enqueuing and dequeuing processes. These two allocations base on QoS features of UMTS traffic to divide a physical queuing buffer into several logical queuing buffers; each logical queuing buffer is used to enqueue arrival UMTS packets. In the DQB allocation, four logical queuing buffers are corresponding to four types of UMTS traffic. And, each logical queuing buffer can be subdivided into a guaranteed buffer and a dynamic buffer; a guaranteed buffer is one type of guaranteed buffers and a dynamic buffer is one type of shared buffers. In the OQB allocation, five logical queuing buffers are used; four logical queuing buffers are guaranteed buffers which are corresponding to four types of UMTS traffic and one overflow logical queuing buffer is a shared buffer. Usually, a logical queuing buffer with higher packet transmission priority can allow to enqueue more packets be enqueued with the DQB and OQB allocations.

In the proposed queuing scheme, a priority-based packet enqueuing module is adopted to process an arrival packet enqueuing job with either the DQB allocation or the OQB allocation. Since two types of queuing buffers, guaranteed buffer and shared buffer, are used to enqueue arrival packets; FIFO scheme is used in a guaranteed buffer and RED-based scheme is used in a shared buffer. The priority-based packet enqueuing module bases on packet transmission priorities of four types of UMTS traffic to assign parameter settings to logical queuing buffers in a differentiated way. With the differentiated parameter settings, the proposed packet enqueuing module can support a differentiated packet enqueuing behavior among UMTS traffic. Moreover, a WRR-based packet dequeuing module is used to manipulate a packet dequeuing process among logical queuing buffers. The SWRR scheme is proposed in the packet dequeuing module; it bases on packet transmission priorities of UMTS packets which are enqueued in logical queuing buffers to assign differentiated packet dequeuing turns to logical queuing buffers. According to received packet dequeuing turns, it is easier for the SWRR scheme to dequeue UMTS packets with higher packet transmission priorities from logical queuing buffers. A differentiated packet dequeuing process can be supported by the SWRR scheme.

In this study, several C++ programs and TCL scripts are coded and implemented in the ns2 to simulate several scenarios. Two types of scenarios, continuous and intermittent transmission patterns, are simulated to understand packet forwarding performance of the proposed queuing scheme with the DQB and OQB allocations. According to the simulation results, we can find several important points. First, the proposed queuing scheme can support four types of UMTS traffic to receive their deserved packet forwarding performance either with the DQB allocation or with the OQB allocation in both a continuous transmission pattern and an intermittent transmission pattern. Secondly, examining the simulation results, we can find that the overall packet

forwarding performance of UMTS traffic which is supported by the proposed queuing scheme either with the DQB allocation or with the OQB allocation is close. Different types of UMTS traffic receive better packet forwarding performance either with the DQB allocation or with the OQB allocation in different scenarios; neither the DQB allocation nor with the OQB allocation can let four types of UMTS traffic receive better packet forwarding performance in all scenarios. Thirdly, about the two queuing buffer allocations, some issues need to be refined. For the DQB allocation, UMTS traffic with the lowest packet transmission priority might receive a packet forwarding starvation in a continuous transmission pattern; but this issue seems not exist in an intermittent transmission pattern. For the OQB allocation, disorder transmission packets occur among four types of UMTS traffic in both a continuous transmission pattern and an intermittent transmission pattern; this issue results from the overflow logical queuing buffer operation in the SWRR scheme.



References

1. UMTSWorld.com , 3G Tutorial, Overview of The Universal Mobile Telecommunication System (draft), available at http://www.umtsworld.com/technology/overview.htm, Jul. 2002.
2. 3GPP TS 23.101, General Universal Mobile Telecommunications System (UMTS) architecture, V9.0.0, Dec. 2009.
3. 3GPP TR 23.922, Architecture for an All IP network, Oct. 1999.
4. UMTS Forum, Enabling UMTS Third Generation Services and Applications (Report 11), available at http://www.tik.ee.ethz.ch/~mobydick/related_work/umts-forum/umts-forum_report11.pdf, Oct. 2000.
5. Heikki Kaaranen, Ari Ahtiainen, Lauri Laitinen, Siammak Naghian and Valtteri Niemi, UMTS Networks: Architecture, Mobility and Services (2nd ed.), *John Wiley & Sons, Inc.*, New Jersey, 2005.
6. 3GPP TS 23.002, Network Architecture, V8.3.0, Sep. 2008.
7. ETSI TS 123 002, Digital cellular telecommunications system (Phase 2+); Universal Mobile. Telecommunications System (UMTS); Network Architecture, V7.1.0, Mar. 2006.
8. Farshid Agharebparast and Victor C. M. Leung, QoS Support in the UMTS/GPRS Backbone Network Using DiffServ, *Proc. of IEEE GLOBECOM*, Nov. 2002.
9. Daniel Collins and Clint Smith, 3G Wireless Networks, *McGraw-Hill Professional*, Sep. 18, 2001.
10. 3GPP, TS 23.207 v8.0.0, End-to-End QoS Concept and Architecture, Dec. 2008.
11. Neal Seitz, ITU-T QoS standards for IP-based networks, *IEEE Communication Magazine*, Vol. 41, No. 6, pp. 82-89, Jun. 2003.
12. Alan Clark, Packet loss Distributions and Packet loss Models, *ITU-T Contribution COM* 12-D97-E, ITU, Telchemy Incorporated 2003.
13. Wenyu Jiang and Henning Schulzrinne, Modeling of Packet Loss

and Delay and their Effect on Real-Time Multimedia Service Quality, *In ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Chapel Hill, North Carolina, Jun. 2000.

14. Thomas Pliakas, George Kormentzas and Charalabos Skianis, Scalable Video Streaming Traffic Delivery in IP/UMTS Networking Environments, *Journal of Multimedia*, Vol. 2, No. 2, pp. 37-46, Apr. 2007.
15. David J. Goodman and Robert A. Myers, Mobile Video Telephony: for 3G Wireless Networks, *McGraw-Hill Professional*, Nov. 2004.
16. 3GPP TS 23.107 V6.4.0, QoS Concept and Architecture, Release 6, Mar. 2006.
17. ETSI TS 122 105, V6.4.0, UMTS; Services and Service Capabilities, available at http://www.etsi.org, Sep. 2005.
18. Robert Lloyd Evans, QoS in Integrated 3G Networks, *Artech House*, New York, 2002.
19. Santiago Alvarez, QoS for IP/MPLS Networks, *Cisco Press*, July 2006.
20. Dimitrios Stiliadis and Anujan Varma, Efficient fair queueing algorithm for packet-switched networks, *IEEE/ACM Transactions on Networking*, Vol. 6, pp. 175-185, Apr. 1998.
21. IETF Differentiated Services (DiffServ) working group, available at: http://www.ietf.org/html.charters/diffserv-charter.html
22. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, An Architecture for Differentiated Services, *IETF RFC 2475*, Dec. 1998.
23. Paul P. White, RSVP and integrated services in the internet: A tutorial, *IEEE Communications Magazine*, Vol. 35, No.5, pp. 100-106, May 1997.
24. Dovrolis Constantinos and Ramanathan Parameswaran, A case for relative differentiated services and the proportional differentiation

model, *IEEE Network*, Vol. 13, pp. 26-34, Sep./Oct. 1999.

25. Tham Chen Khong, Yao Qi, and Jiang Yuming, Achieving differentiated services through multi-class probabilistic priority scheduling, *Computer Networks*, Vol.40, No. 4, pp. 577-593, 2002.
26. Y. Bernet, J. Binder, S. Blake, M. Carlson, S. Keshav, E. Davies, B. Ohlman, D. Verma, Z. Wang and W. Weiss, A Framework for Differentiated Service, *Internet Draft*, available at http://tools.ietf.org/id/draft-ietf-diffserv-framework-01.txt, Oct. 1998.
27. Ikjun Yeom and A. L. Narasimha Reddy, Modeling TCP Behavior in a Differentiated Services Network, *IEEE/ACM Transactions on Networking*, pp. 31-46, Vol. 9, No. 1, Feb. 2001.
28. Van Jacobson, Kathleen Nichols and Kedar Poduri, An expedited forwarding PHB. Request for Comments 2598, *Internet Engineering Task Force*, Jun. 1999.
29. Dana Arash and Malekloo Ahmad, Performance Comparison between Active and Passive Queue Management, *IJCSI International Journal of Computer Science Issues*, Vol. 7, Issue 3, No. 5, May 2010.
30. Bob Braden, David Clark, Jon Crowcroft, Bruce Davie, Steve Deering, Deborah Estrin, Sally Floyd, Van Jacobson, Greg Minshall, Craig Partridge, Larry Peterson, Kadangode Ramakrishnan, Scott Shenker, John Wroclwski, and Lixia Zhang, Recommendations on Queue Management and Congestion Avoidance in the Internet, *IETF RFC 2309*, Apr. 1998.
31. Sally Floyd, References on RED (Random Early Detection) Queue Management, available at http://www.icir.org/floyd/red.html
32. James Aweya, Michel Ouellette and Delfin Y. Montuno, A multi-queue TCP window control scheme with dynamic buffer allocation, *Journal of Systems Architecture*, Vol. 49, No. 7-9, pp. 369-385, Oct. 2003.

33. Sally Floyd, RED: discussions of setting parameters, available at http://www.icir.org/floyd/REDparameters.txt, Nov. 1997.
34. Thomas Ziegler, Christof Brandauer and Serge Fdida, A Quantitative Model for the Parameter Setting of RED with TCP Traffic, *IWQoS, LNCS 2092*, Vol. 2092, pp. 202-216, 2001.
35. David Clark and Wenjia Fang, Explicit Allocation of Best Effort Packet Delivery Service, *IEEE/ACM Transactions on Networking*, Vol. 6, No. 4, pp. 362-373, Aug. 1998.
36. Dong Lin and Robert Morris, Dynamic of Random Early Detection, *In Proceedings of ACM SIGCOMM97*, pp. 127-137, Cannes, France, Sep. 14-18, 1997.
37. Jaiswal N.K., Priority Queue, *Academic Press*, Net York, 1968.
38. Eitan Altman and Tania Jiménez, Simulation analysis of RED with short lived TCP connections, *Computer Networks*, Vol. 44, Issue 5, pp. 631-641, Apr. 2004.
39. Mohamed Ashour and Tho Le-Ngoc, Priority queuing of long-range dependent traffic, *In Proceedings of the 2004 IEEE Global Telecommunications Conference (GLOBECOM'04)*, Vol. 6, pp. 3025-3029, 2003.
40. Hoon Lee, Anatomy of delay performance for the strict priority scheduling scheme in multi-service Internet, *Computer Communications*, Vol. 29, No. 1, pp. 69-76, 1 Dec. 2005.
41. Chung G. Kang and Harry H. Tan, Queueing analysis of explicit priority assignment buffer access scheme for ATM networks, *Computer Communications*, Vol. 21, No. 11, pp. 996-1009, 10 Aug. 1998.
42. Takahiro Matsuda, Akira Nagata, and Miki Yamaoto, Performance Analysis and Improvement of HighSpeed TCP with TailDrop/RED Routers, *IEICE Transactions on Communications*, Vol.E88–B, No.6 , pp. 2495-2507, Jun. 2005.
43. Long Bao Le, Ekram Hossain and Attahiru Sule Alfa, Service

differentiation in multirate wireless networks with weighted round-robin scheduling and ARQ-based error control, *IEEE Transactions on Communications*, vol.54, No.2, pp.208–215, Feb. 2006.

44. Christoph Lindemann and Axel Thummler, Evaluating the GPRS radio interface for different quality of service profiles, *Proceeding 12th GI/ITG Fachtagung Kommunikation in Verteilten System*, pp. 291-301, Hamburg, Germany, Feb. 2001.
45. Robert Braden, David Clark, and Scott Shenker, Integrated Services in the Internet Architecture: an Overview, *IETF RFC 163*3, Jul. 1994.
46. Sally Floyd and Van Jacobson, Random Early Detection Gateways for Congestion Avoidance, *IEEE/ACM Transactions on Networking*, Vol 1, No. 4, pp. 397-413, Aug. 1993.
47. Fengyuan Rena, Chuang Lina and Bo Wei, A nonlinear control theoretic analysis to TCP–RED system, *Computer Networks*, Vol. 49, Issue 4, No.15, pp. 580-592, Nov. 2005.
48. Chait Hollot, Vishal Misra, Don Towsley and Wei-Bo Gong, A control theoretic analysis of RED, *IEEE Infocom'01, Anchorage*, Alaska, USA, 2001.
49. Juha Heinanen and Kalevi Kilkki, A fair buffer allocation scheme, *Computer Communications*, Vol. 21, No. 3, pp. 220-226, 25 Mar. 1998
50. Gianluca Mazzini, Riccardo Rovatti and Gianluca Setti, A closed from solution of Bernoullian two-classes priority queue, *Computer Communications*, vol. 9, No. 3, pp. 264-266, 2005.
51. Xiao Yang, Performance analysis of priority schemes for IEEE 802.11 and IEEE 802.11e wireless LANs, *IEEE Transactions on Wireless Communications*, Vol.4, No.4, pp. 1506-1515, 2005.
52. Steven McCanne and Sally Floyd, The Network Simulator – ns2, available at: http://nsnam.isi.edu/nsnam/index.php/Main_Page

Appendix

Appendix A: The DQB Source Code

```
#include "umtspred-queue.h"
static class UmtsPredQueueClass : public TclClass
{
public:
            UmtsPredQueueClass() : TclClass("Queue/UmtsPredQueue") {}
            TclObject* create(int, const char*const*) {
                return (new UmtsPredQueue);
  }
} class_umts_priority_red_round_robin;

UmtsPredQueue::UmtsPredQueue()
{
   con_q_ = new PacketQueue;
   str_q_ = new PacketQueue;
   int_q_ = new PacketQueue;
   bac_q_ = new PacketQueue;
   pq_ = con_q_;
   con_enable_enque_ = false;
   str_enable_enque_ = false;
   int_enable_enque_ = false;
   bac_enable_enque_ = false;
   packet_deque_ctr_ = 0;
   umts_deq_turn_ = 0;
   phy_q_length_ = 0;
   con_enq_ctr_ = 0;
   str_enq_ctr_ = 0;
   int_enq_ctr_ = 0;
   bac_enq_ctr_ = 0;
   con_deq_ctr_ = 0;
   str_deq_ctr_ = 0;
   int_deq_ctr_ = 0;
   bac_deq_ctr_ = 0;
   con_dpq_ctr_ = 0;
   str_dpq_ctr_ = 0;
   int_dpq_ctr_ = 0;
   bac_dpq_ctr_ = 0;

   bind("con_queue_limit", &con_queue_limit_);
   bind("str_queue_limit", &str_queue_limit_);
   bind("int_queue_limit", &int_queue_limit_);
   bind("bac_queue_limit", &bac_queue_limit_);
   bind("c_enq_min", &c_enq_min_);
   bind("s_enq_min", &s_enq_min_);
   bind("i_enq_min", &i_enq_min_);
   bind("b_enq_min", &b_enq_min_);
   bind("c_enq_max", &c_enq_max_);
   bind("s_enq_max", &s_enq_max_);
   bind("i_enq_max", &i_enq_max_);
   bind("b_enq_max", &b_enq_max_);
   bind("con_redenque_prob", &con_redenque_prob_);
   bind("str_redenque_prob", &str_redenque_prob_);
   bind("int_redenque_prob", &int_redenque_prob_);
   bind("bac_redenque_prob", &bac_redenque_prob_);
}

void UmtsPredQueue::enque(Packet* p)
{
   hdr_ip* iph = hdr_ip::access(p);
     phy_q_length_ = con_q_ -> length() + str_q_ -> length() + int_q_ -> length() + bac_q_->length();
   switch (iph->prio_)
   {
    // If IPv6 priority = 15 (UMTS conversation traffic IP packet) then enqueue to con-queue
    case 15:
```

```
      {
       if (phy_q_length_ < qlim_)
{
            if (con_q_->length() < con_queue_limit_)
               {
                con_enable_enque_ = true;
            } else {
                        con_enable_enque_ =
                         overflow_pkt_enqueue_by_red(c_enq_min_,c_enq_max_,phy_q_length_,con_red
                         enque_prob_);
               }
       } else {
                con_enable_enque_ =    false;
          }
       if (con_enable_enque_)
          {
           con_q_->enque(p);
  con_enq_ctr_++;
       } else {
                con_dpq_ctr_++;
          }
       break;
}
// If IPv6 priority = 13 (UMTS stream traffic IP packet) then enqueue to str-queue
case 13:
      {
       if (phy_q_length_ < qlim_)
{
            if (str_q_->length() < str_queue_limit_)
               {
      str_enable_enque_ = true;
            } else {
                 str_enable_enque_ =
                 overflow_pkt_enqueue_by_red(s_enq_min_,s_enq_max_,phy_q_length_,str_redenqu
                 e_prob_);
               }
       } else {
                str_enable_enque_ =    false;
          }
       if (str_enable_enque_)
          {
           str_q_->enque(p);
  str_enq_ctr_++;
       } else {
                str_dpq_ctr_++;
          }
       break;
}
// If IPv6 priority = 11 (UMTS interactive traffic IP packet) then enqueue to int-queue
case 11:
      {
       if (phy_q_length_ < qlim_)
{
            if (int_q_->length() < int_queue_limit_)
               {
      int_enable_enque_ = true;
            } else {
                        int_enable_enque_ =
                         overflow_pkt_enqueue_by_red(i_enq_min_,i_enq_max_,phy_q_length_,int_reden
                         que_prob_);
               }
       } else {
                int_enable_enque_ =    false;
          }
       if (int_enable_enque_)
```

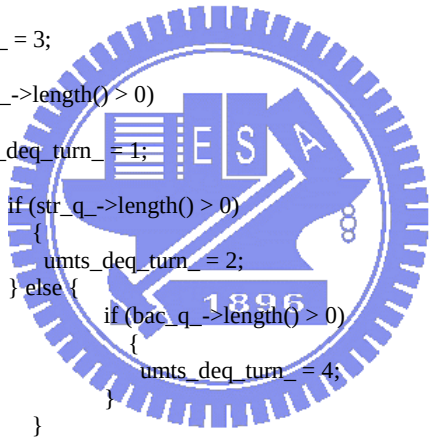
```
                {
                  int_q_->enque(p);
        int_enq_ctr_++;
                } else {
                        int_dpq_ctr_++;
                }
                break;
        }
        // If IPv6 priority = 9 (UMTS background traffic IP packet) then enqueue to bac-queue
        case 9:
              {
                if (phy_q_length_ < qlim_)
        {
                    if (bac_q_->length() < bac_queue_limit_)
                       {
                bac_enable_enque_ = true;
                    } else {
                                bac_enable_enque_ =
                                 overflow_pkt_enqueue_by_red(b_enq_min_,b_enq_max_,phy_q_length_,bac_red
                                 enque_prob_);
                       }
                } else {
                        bac_enable_enque_ =    false;
                   }
                if (bac_enable_enque_)
                   {
                    bac_q_->enque(p);
        bac_enq_ctr_++;
                } else {
                        bac_dpq_ctr_++;
                   }
                break;
        }
        default:
              {
                 printf("Not an UMTS traffic packet!! Be dropped by UmtsPred queue.\n");
                break;
        }
       }

}
unsigned long UmtsPredQueue::random_seed_by_system_time()
{
  struct timeval tv;
  struct timezone tz;
  unsigned long rng_seed;
/* to generate a seed of random generator by timer */
  if (gettimeofday(&tv, &tz) == 0)
     {
       rng_seed=tv.tv_usec;
  } else {
              printf("Not get timer information by gettimeofday !!!\n");
              rng_seed=0;
     }

  return rng_seed;

}

   bool UmtsPredQueue::overflow_pkt_enqueue_by_red(int enq_min, int enq_max, int enq_pkt_num, double
      redenque_prob)
{
  unsigned long rseed;
  double random_prob;
```

```
 if (enq_pkt_num >= enq_max)
    {
      return false;
 } else {
              if (enq_pkt_num < enq_min)
                 {
                   return true;
              } else {
                           rseed=random_seed_by_system_time();
                           srand(((int)rseed));
                           random_prob=(double)(rand()/(RAND_MAX+1.0));
                           if (random_prob <= redenque_prob)
                              {
                                return true;
                           } else {
                                        return false;
                              }
                 }
    }

}

Packet* UmtsPredQueue::deque()
{
   Packet *p;
   bool deque_packet = false;

   umts_deq_turn_ = 0;
   switch (packet_deque_ctr_ % 10)
   {
    // to set dequeue turn to conversation traffic first if con_q_->length()    //case (0, 3, 6, 9):
    case 0:
          {
 if (con_q_->length() > 0)
               {
                 umts_deq_turn_ = 1;
            } else {
                        if (str_q_->length() > 0)
                           {
                             umts_deq_turn_ = 2;
                        } else {
                                    if (int_q_->length() > 0)
                                       {
                                         umts_deq_turn_ = 3;
                                    } else {
                                                if (bac_q_->length() > 0)
                                                   {
                                                     umts_deq_turn_ = 4;
                                                }
                                       }
                           }
               }
            break;
    }
    case 3:
          {
 if (con_q_->length() > 0)
               {
                 umts_deq_turn_ = 1;
            } else {
                        if (str_q_->length() > 0)
                           {
                             umts_deq_turn_ = 2;
                        } else {
                                    if (int_q_->length() > 0)
```

```
                                {
                                    umts_deq_turn_ = 3;
                                } else {
                                        if (bac_q_->length() > 0)
                                            {
                                                umts_deq_turn_ = 4;
                                            }
                                    }
                            }
                    }
                break;
    }
    case 6:
            {
if (con_q_->length() > 0)
                    {
                        umts_deq_turn_ = 1;
                } else {
                            if (str_q_->length() > 0)
                                {
                                    umts_deq_turn_ = 2;
                                } else {
                                        if (int_q_->length() > 0)
                                            {
                                                umts_deq_turn_ = 3;
                                            } else {
                                                    if (bac_q_->length() > 0)
                                                        {
                                                            umts_deq_turn_ = 4;
                                                        }
                                                }
                                    }
                    }
                break;
    }
    case 9:
            {
if (con_q_->length() > 0)
                    {
                        umts_deq_turn_ = 1;
                } else {
                            if (str_q_->length() > 0)
                                {
                                    umts_deq_turn_ = 2;
                                } else {
                                        if (int_q_->length() > 0)
                                            {
                                                umts_deq_turn_ = 3;
                                            } else {
                                                    if (bac_q_->length() > 0)
                                                        {
                                                            umts_deq_turn_ = 4;
                                                        }
                                                }
                                    }
                    }
                break;
    }
    // to set dequeue turn to stream traffic first if con_q_->length()      //case (1, 4, 7):
    case 1:
            {
if (str_q_->length() > 0)
                    {
                        umts_deq_turn_ = 2;
                } else {
```

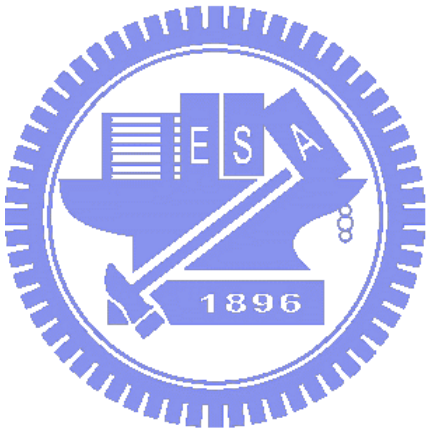
```
                    if (con_q_->length() > 0)
                       {
                        umts_deq_turn_ = 1;
                    } else {
                            if (int_q_->length() > 0)
                               {
                                umts_deq_turn_ = 3;
                            } else {
                                    if (bac_q_->length() > 0)
                                       {
                                        umts_deq_turn_ = 4;
                                    }
                               }
                       }
                }
            break;
    }
    case 4:
           {
if (str_q_->length() > 0)
                {
                 umts_deq_turn_ = 2;
            } else {
                    if (con_q_->length() > 0)
                       {
                        umts_deq_turn_ = 1;
                    } else {
                            if (int_q_->length() > 0)
                               {
                                umts_deq_turn_ = 3;
                            } else {
                                    if (bac_q_->length() > 0)
                                       {
                                        umts_deq_turn_ = 4;
                                    }
                               }
                       }
                }
            break;
    }
    case 7:
           {
if (str_q_->length() > 0)
                {
                 umts_deq_turn_ = 2;
            } else {
                    if (con_q_->length() > 0)
                       {
                        umts_deq_turn_ = 1;
                    } else {
                            if (int_q_->length() > 0)
                               {
                                umts_deq_turn_ = 3;
                            } else {
                                    if (bac_q_->length() > 0)
                                       {
                                        umts_deq_turn_ = 4;
                                    }
                               }
                       }
                }
            break;
    }
    // to set dequeue turn to interactive traffic first if con_q_->length()    //case (2, 8):
    case 2:
```

```
        {
if (int_q_->length() > 0)
            {
              umts_deq_turn_ = 3;
          } else {
                    if (con_q_->length() > 0)
                       {
                         umts_deq_turn_ = 1;
                    } else {
                              if (str_q_->length() > 0)
                                 {
                                   umts_deq_turn_ = 2;
                              } else {
                                        if (bac_q_->length() > 0)
                                           {
                                             umts_deq_turn_ = 4;
                                        }
                                   }
                         }
              }
          break;
   }
   case 8:
        {
if (int_q_->length() > 0)
            {
              umts_deq_turn_ = 3;
          } else {
                    if (con_q_->length() > 0)
                       {
                         umts_deq_turn_ = 1;
                    } else {
                              if (str_q_->length() > 0)
                                 {
                                   umts_deq_turn_ = 2;
                              } else {
                                        if (bac_q_->length() > 0)
                                           {
                                             umts_deq_turn_ = 4;
                                        }
                                   }
                         }
              }
          break;
   }
   // to set dequeue turn to background traffic first if con_q_->length()      //case (5):
   case 5:
        {
if (bac_q_->length() > 0)
            {
              umts_deq_turn_ = 4;
          } else {
                    if (con_q_->length() > 0)
                       {
                         umts_deq_turn_ = 1;
                    } else {
                              if (str_q_->length() > 0)
                                 {
                                   umts_deq_turn_ = 2;
                              } else {
                                        if (int_q_->length() > 0)
                                           {
                                             umts_deq_turn_ = 3;
                                        }
                                   }
                         }
```



```
                        }
                    }
                break;
    }
    default:
            {
             umts_deq_turn_ = 0;
             break;
    }
   }

   switch(umts_deq_turn_)
   {
     case 1:
            {
             p = con_q_->deque();
             deque_packet = true;
  con_deq_ctr_++;
             break;
    }
    case 2:
            {
             p = str_q_->deque();
             deque_packet = true;
  str_deq_ctr_++;
             break;
    }
    case 3:
            {
             p = int_q_->deque();
             deque_packet = true;
  int_deq_ctr_++;
             break;
    }
    case 4:
            {
             p = bac_q_->deque();
             deque_packet = true;
  bac_deq_ctr_++;
             break;
    }
    default:
            {
             break;
    }
   }
   if (deque_packet == true)
      {
       packet_deque_ctr_++;
       return (p);
   } else
            { return 0;}
}

// To print out the statistics information of UMTS priority queueing discipling    // Unit : packet

void UmtsPredQueue::printstats()
{
  printf("\n***    UMTS priority queueing discipline operation statistics    ***\n");
  printf("=================================================\n");
printf("Traffic type              enqueue                dequeue                dropped\n");
printf("------------              -------                -------                -------\n");
   printf("Conversation              %7d                %7d
      %7d\n",con_enq_ctr_,con_deq_ctr_,con_dpq_ctr_);
   printf("Stream                    %7d                %7d
      %7d\n",str_enq_ctr_,str_deq_ctr_,str_dpq_ctr_);
```

```
  printf("Interactive                %7d                    %7d
     %7d\n",int_enq_ctr_,int_deq_ctr_,int_dpq_ctr_);
  printf("Background                 %7d                    %7d
     %7d\n",bac_enq_ctr_,bac_deq_ctr_,bac_dpq_ctr_);
printf("\n---------------------------------------------------------------------\n");
printf("Total UMTS dequeue packet number : %d\n",packet_deque_ctr_);

}
// Commands from the ns file are interpreted through this interface.
int UmtsPredQueue::command(int argc, const char*const* argv)
{
if (strcmp(argv[1], "printstats") == 0)
{
       printstats();
       return (TCL_OK);
   }
 return(Queue::command(argc, argv));

}
```



Appendix B: The DQB Header File

```
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include "queue.h"
#include "address.h"

class UmtsPredQueue : public Queue {
  public:
UmtsPredQueue();
int command(int argc, const char*const* argv);        // interface to ns scripts
protected:
void enque(Packet*);
unsigned long random_seed_by_system_time();
   bool overflow_pkt_enqueue_by_red(int enq_min, int enq_max, int enq_pkt_num, double redenque_prob);

   Packet* deque();
   void printstats();
PacketQueue *con_q_;        // the FIFO queue for UMTS conversation traffic
PacketQueue *str_q_;        // the FIFO queue for UMTS stream traffic
PacketQueue *int_q_;        // the FIFO queue for UMTS interactive traffic
PacketQueue *bac_q_;        // the FIFO queue for UMTS background traffic
bool con_enable_enque_;    // boolean for UMTS conversational packet enqueue
bool str_enable_enque_;    // boolean for UMTS streaming packet enqueue
bool int_enable_enque_;    // boolean for UMTS interactive packet enqueue
bool bac_enable_enque_;    // boolean for UMTS background packet enqueue
int packet_deque_ctr_;     // to count the sent packets
int pkt_deque_cycle_;      // to mod the sent packets
int umts_deq_turn_;            // 1 for con_q, 2 for str_q,   // 3 for int_q, 4 for bac_q
int phy_q_length_;           // used queue buffer length
 int con_enq_ctr_;             // UMTS conversation application enqueue packet counter
int str_enq_ctr_;          // UMTS stream application enqueue packet counter
int int_enq_ctr_;          // UMTS interactive application enqueue packet counter
int bac_enq_ctr_;    // UMTS background application enqueue packet counter
int con_deq_ctr_;       // UMTS conversation application dequeue packet counter
int str_deq_ctr_;          // UMTS stream application dequeue packet counter
int int_deq_ctr_;          // UMTS interactive application dequeue packet counter
int bac_deq_ctr_;       // UMTS background application dequeue packet counter
int con_dpq_ctr_;          // UMTS conversation application packet drop by queue
int str_dpq_ctr_;          // UMTS stream application packet drop by queue
int int_dpq_ctr_;          // UMTS interactive application packet drop by queue
int bac_dpq_ctr_;          // UMTS background application packet drop by queue
int con_queue_limit_;        // UMTS conversational application queue limit
int str_queue_limit_;       // UMTS streaming application queue limit
int int_queue_limit_;       // UMTS interactive application queue limit
int bac_queue_limit_;        // UMTS background application queue limit
int c_enq_min_;              // UMTS conversational application min queue limit
int c_enq_max_;           // UMTS conversation application max queue limit
int s_enq_min_;           // UMTS streaming application min queue limit
int s_enq_max_;           // UMTS streaming application max queue limit
int i_enq_min_;           // UMTS interactive application min queue limit
int i_enq_max_;           // UMTS interactive application max queue limit
int b_enq_min_;           // UMTS background application min queue limit
int b_enq_max_;           // UMTS background application max queue limit

double con_redenque_prob_;      //UMTS conversational aplication packet enqueue probability by red scheme
double str_redenque_prob_;       //UMTS streaming aplication packet enqueue probability by red scheme
double int_redenque_prob_;       //UMTS interactive aplication packet enqueue probability by red scheme
double bac_redenque_prob_;        //UMTS background aplication packet enqueue probability by red scheme
};
```

Appendix C: The OQB Source code

```
#include "umtspdb-queue.h"

static class UmtsPdbQueueClass : public TclClass {
public:
            UmtsPdbQueueClass() : TclClass("Queue/UmtsPdbQueue") {}
            TclObject* create(int, const char*const*) {
           return (new UmtsPdbQueue);
  }
} class_umts_priority_dynamic_overflow_buffer;


UmtsPdbQueue::UmtsPdbQueue()
{
   FILE *fp;

   con_q_ = new PacketQueue;
   str_q_ = new PacketQueue;
   int_q_ = new PacketQueue;
   bac_q_ = new PacketQueue;
   ovb_q_ = new PacketQueue;
   pq_ = con_q_;
   con_enable_enque_ = false;
   str_enable_enque_ = false;
   int_enable_enque_ = false;
   bac_enable_enque_ = false;
   ovb_enable_enque_ = false;
   packet_deque_ctr_ = 0;
   umts_deq_turn_ = 0;
   phy_q_length_ = 0;
   con_enq_ctr_ = 0;
   str_enq_ctr_ = 0;
   int_enq_ctr_ = 0;
   bac_enq_ctr_ = 0;
   ovb_enq_ctr_ = 0;
   con_deq_ctr_ = 0;
   str_deq_ctr_ = 0;
   int_deq_ctr_ = 0;
   bac_deq_ctr_ = 0;
   ovb_deq_ctr_ = 0;
   con_dpq_ctr_ = 0;
   str_dpq_ctr_ = 0;
   int_dpq_ctr_ = 0;
   bac_dpq_ctr_ = 0;

  bind("con_queue_limit", &con_queue_limit_);
   bind("str_queue_limit", &str_queue_limit_);
   bind("int_queue_limit", &int_queue_limit_);
   bind("bac_queue_limit", &bac_queue_limit_);
   bind("c_enq_min", &c_enq_min_);
   bind("s_enq_min", &s_enq_min_);
   bind("i_enq_min", &i_enq_min_);
   bind("b_enq_min", &b_enq_min_);
   bind("c_enq_max", &c_enq_max_);
   bind("s_enq_max", &s_enq_max_);
   bind("i_enq_max", &i_enq_max_);
   bind("b_enq_max", &b_enq_max_);
   bind("con_dbenque_prob", &con_dbenque_prob_);
   bind("str_dbenque_prob", &str_dbenque_prob_);
   bind("int_dbenque_prob", &int_dbenque_prob_);
   bind("bac_dbenque_prob", &bac_dbenque_prob_);

   /* To check output record files and remove them */
```

```
   if ((fp = fopen("con_pkt_forwarding_delay_rec_pdb","w")) != NULL)
      {
       fclose(fp);
   }
   if ((fp = fopen("str_pkt_forwarding_delay_rec_pdb","w")) != NULL)
      {
       fclose(fp);
   }
   if ((fp = fopen("int_pkt_forwarding_delay_rec_pdb","w")) != NULL)
      {
       fclose(fp);
   }
   if ((fp = fopen("bac_pkt_forwarding_delay_rec_pdb","w")) != NULL)
      {
       fclose(fp);
   }

}

void UmtsPdbQueue::enque(Packet* p)
{
   hdr_ip* iph = hdr_ip::access(p);

    phy_q_length_ = con_q_->length() + str_q_->length() + int_q_->length() + bac_q_->length() +
     ovb_q_->length();
   switch (iph->prio_)
   {
    // If IPv6 priority = 15 (UMTS conversation traffic IP packet) then enqueue to con-queue
    case 15:
          {
           if (phy_q_length_ < qlim_)
    {
                if (con_q_->length() < con_queue_limit_)
                   {
                    con_enable_enque_ = true;
                } else {
                           con_enable_enque_ = false;
                             ovb_enable_enque_ =
                              overflow_pkt_enqueue_by_prob(c_enq_min_,c_enq_max_,phy_q_length_,con_db
                              enque_prob_);
                   }
           } else {
                      con_enable_enque_ = false;
                      ovb_enable_enque_ = false;
              }
           if (con_enable_enque_)
              {
                 con_q_->enque(p);
                 con_enq_ctr_++;
           } else {
                      if (ovb_enable_enque_)
                         {
                                ovb_q_->enque(p);
                                con_enq_ctr_++;
                                ovb_enq_ctr_++;
                      } else {
                                  con_dpq_ctr_++;
                           }
              }
           break;
    }

    // If IPv6 priority = 13 (UMTS stream traffic IP packet) then enqueue to str-queue
    case 13:
          {
           if (phy_q_length_ < qlim_)
```

```
{
            if (str_q_->length() < str_queue_limit_)
                {
                 str_enable_enque_ = true;
                } else {
                            str_enable_enque_ = false;
                              ovb_enable_enque_ =
                               overflow_pkt_enqueue_by_prob(s_enq_min_,s_enq_max_,phy_q_length_,str_dbe
                               nque_prob_);
                }
        } else {
                    str_enable_enque_ = false;
                    ovb_enable_enque_ = false;
            }
        if (str_enable_enque_)
            {
               str_q_->enque(p);
               str_enq_ctr_++;
        } else {
                    if (ovb_enable_enque_)
                       {
                              ovb_q_->enque(p);
                              str_enq_ctr_++;
                              ovb_enq_ctr_++;
                    } else {
                                 str_dpq_ctr_++;
                       }
            }
        break;
}
// If IPv6 priority = 11 (UMTS interactive traffic IP packet) then enqueue to int-queue
case 11:
       {
        if (phy_q_length_ < qlim_)
{
            if (int_q_->length() < int_queue_limit_)
                {
                 int_enable_enque_ = true;
                } else {
                            int_enable_enque_ = false;
                              ovb_enable_enque_ =
                               overflow_pkt_enqueue_by_prob(i_enq_min_,i_enq_max_,phy_q_length_,int_dbe
                               nque_prob_);
                }
        } else {
                    int_enable_enque_ = false;
                    ovb_enable_enque_ = false;
            }
        if (int_enable_enque_)
            {
              int_q_->enque(p);
  int_enq_ctr_++;
        } else {
                    if (ovb_enable_enque_)
                       {
                              ovb_q_->enque(p);
                              int_enq_ctr_++;
                              ovb_enq_ctr_++;
                    } else {
                                 int_dpq_ctr_++;
                       }
            }
        break;
}
```

```
    // If IPv6 priority = 9 (UMTS background traffic IP packet) then enqueue to bac-queue
    case 9:
          {
           if (phy_q_length_ < qlim_)
    {
                if (bac_q_->length() < bac_queue_limit_)
                   {
                    bac_enable_enque_ = true;
                } else {
                           bac_enable_enque_ = false;
                             ovb_enable_enque_ =
                              overflow_pkt_enqueue_by_prob(b_enq_min_,b_enq_max_,phy_q_length_,bac_d
                              benque_prob_);
                   }
           } else {
                       bac_enable_enque_ = false;
                       ovb_enable_enque_ = false;
              }
           if (bac_enable_enque_)
              {
                 bac_q_->enque(p);
                 bac_enq_ctr_++;
           } else {
                       if (ovb_enable_enque_)
                          {
                                 ovb_q_->enque(p);
                                 bac_enq_ctr_++;
                                 ovb_enq_ctr_++;
                       } else {
                                 bac_dpq_ctr_++;
                          }
              }
           break;
    }
    default:
          {

  // printf("Not an UMTS traffic packet!! Be dropped by UmtsPi queue.\n");
           break;
    }
   }

}

unsigned long UmtsPdbQueue::random_seed_by_system_time()
{
 struct timeval tv;
 struct timezone tz;
 unsigned long rng_seed;

/* to generate a seed of random generator by timer */
 if (gettimeofday(&tv, &tz) == 0)
    {
      rng_seed=tv.tv_usec;
 } else {
             printf("Not get timer information by gettimeofday !!!\n");
             rng_seed=0;
    }

 return rng_seed;

}

  bool UmtsPdbQueue::overflow_pkt_enqueue_by_prob(int enq_min, int enq_max, int enq_pkt_num,
   double dbenque_prob)
```

```
{
 unsigned long rseed;
 double random_prob;

 if (enq_pkt_num >= enq_max)
    {
      return false;
 } else {
              if (enq_pkt_num < enq_min)
                 {
                   return true;
              } else {
                           rseed=random_seed_by_system_time();
                           srand(((int)rseed));
                           random_prob=(double)(rand()/(RAND_MAX+1.0));
                           if (random_prob <= dbenque_prob)
                              {
                                return true;
                           } else {
                                        return false;
                              }
                 }
    }

}

Packet* UmtsPdbQueue::deque()
{
   Packet *p;
   bool deque_packet = false;
   int packet_forwarding_cycle = 15;

   umts_deq_turn_ = 0;
   switch (packet_deque_ctr_ % packet_forwarding_cycle)
   {
    // to set dequeue turn to conversation traffic first if con_q_->length()      //case (0, 3, 6, 9, 12):
    case 0:
          {
  if (con_q_->length() > 0)
                {
                  umts_deq_turn_ = 1;
             } else {
                           if (str_q_->length() > 0)
                              {
                                umts_deq_turn_ = 2;
                           } else {
                                        if (int_q_->length() > 0)
                                           {
                                             umts_deq_turn_ = 3;
                                        } else {
                                                     if (bac_q_->length() > 0)
                                                        {
                                                          umts_deq_turn_ = 4;
                                                     } else {
                                                                  if (ovb_q_->length() > 0)
                                                                     {
                                                                       umts_deq_turn_ = 5;
                                                                  }
                                                        }
                                           }
                              }
                 }
          break;
    }
    case 3:
```

```
        {
if (con_q_->length() > 0)
            {
             umts_deq_turn_ = 1;
         } else {
                    if (str_q_->length() > 0)
                       {
                        umts_deq_turn_ = 2;
                    } else {
                                if (int_q_->length() > 0)
                                   {
                                    umts_deq_turn_ = 3;
                                } else {
                                            if (bac_q_->length() > 0)
                                               {
                                                umts_deq_turn_ = 4;
                                            } else {
                                                        if (ovb_q_->length() > 0)
                                                           {
                                                            umts_deq_turn_ = 5;
                                                        }
                                               }
                                   }
                       }
            }
         break;
  }
  case 6:
        {
if (con_q_->length() > 0)
            {
             umts_deq_turn_ = 1;
         } else {
                    if (str_q_->length() > 0)
                       {
                        umts_deq_turn_ = 2;
                    } else {
                                if (int_q_->length() > 0)
                                   {
                                    umts_deq_turn_ = 3;
                                } else {
                                            if (bac_q_->length() > 0)
                                               {
                                                umts_deq_turn_ = 4;
                                            } else {
                                                        if (ovb_q_->length() > 0)
                                                           {
                                                            umts_deq_turn_ = 5;
                                                        }
                                               }
                                   }
                       }
            }
         break;
  }
  case 9:
        {
if (con_q_->length() > 0)
            {
             umts_deq_turn_ = 1;
         } else {
                    if (str_q_->length() > 0)
                       {
                        umts_deq_turn_ = 2;
                    } else {
```

```
                        if (int_q_->length() > 0)
                            {
                             umts_deq_turn_ = 3;
                        } else {
                                if (bac_q_->length() > 0)
                                    {
                                     umts_deq_turn_ = 4;
                                } else {
                                        if (ovb_q_->length() > 0)
                                            {
                                             umts_deq_turn_ = 5;
                                            }
                                    }
                            }
                    }
                }
            break;
    }
    case 12:
            {
if (con_q_->length() > 0)
                {
                 umts_deq_turn_ = 1;
            } else {
                    if (str_q_->length() > 0)
                        {
                         umts_deq_turn_ = 2;
                        } else {
                                if (int_q_->length() > 0)
                                    {
                                     umts_deq_turn_ = 3;
                                } else {
                                        if (bac_q_->length() > 0)
                                            {
                                             umts_deq_turn_ = 4;
                                            } else {
                                                    if (ovb_q_->length() > 0)
                                                        {
                                                         umts_deq_turn_ = 5;
                                                        }
                                            }
                                    }
                            }
                    }
                }
            break;
    }

    // to set dequeue turn to stream traffic first if con_q_->length()      //case (1, 4, 8, 13):
    case 1:
            {
if (str_q_->length() > 0)
                {
                 umts_deq_turn_ = 2;
            } else {
                    if (con_q_->length() > 0)
                        {
                         umts_deq_turn_ = 1;
                        } else {
                                if (int_q_->length() > 0)
                                    {
                                     umts_deq_turn_ = 3;
                                } else {
                                        if (bac_q_->length() > 0)
                                            {
                                             umts_deq_turn_ = 4;
                                            } else {
```

```
                                        if (ovb_q_->length() > 0)
                                            {
                                              umts_deq_turn_ = 5;
                                            }
                                    }
                                }
                            }
                        }
                    break;
    }
    case 4:
            {
if (str_q_->length() > 0)
                {
                  umts_deq_turn_ = 2;
                } else {
                        if (con_q_->length() > 0)
                            {
                              umts_deq_turn_ = 1;
                            } else {
                                    if (int_q_->length() > 0)
                                        {
                                          umts_deq_turn_ = 3;
                                        } else {
                                                if (bac_q_->length() > 0)
                                                    {
                                                      umts_deq_turn_ = 4;
                                                    } else {
                                                            if (ovb_q_->length() > 0)
                                                                {
                                                                  umts_deq_turn_ = 5;
                                                                }
                                                    }
                                        }
                            }
                        }
                    break;
    }
    case 8:
            {
if (str_q_->length() > 0)
                {
                  umts_deq_turn_ = 2;
                } else {
                        if (con_q_->length() > 0)
                            {
                              umts_deq_turn_ = 1;
                            } else {
                                    if (int_q_->length() > 0)
                                        {
                                          umts_deq_turn_ = 3;
                                        } else {
                                                if (bac_q_->length() > 0)
                                                    {
                                                      umts_deq_turn_ = 4;
                                                    } else {
                                                            if (ovb_q_->length() > 0)
                                                                {
                                                                  umts_deq_turn_ = 5;
                                                                }
                                                    }
                                        }
                            }
                        }
                    break;
```

```
  }
  case 13:
        {
if (str_q_->length() > 0)
            {
              umts_deq_turn_ = 2;
         } else {
                    if (con_q_->length() > 0)
                       {
                         umts_deq_turn_ = 1;
                    } else {
                              if (int_q_->length() > 0)
                                 {
                                   umts_deq_turn_ = 3;
                              } else {
                                        if (bac_q_->length() > 0)
                                           {
                                             umts_deq_turn_ = 4;
                                        } else {
                                                  if (ovb_q_->length() > 0)
                                                     {
                                                       umts_deq_turn_ = 5;
                                                  }
                                           }
                                 }
                    }
            }
         break;
  }
  // to set dequeue turn to interactive traffic first if int_q_->length()     //case (2, 10):
  case 2:
        {
if (int_q_->length() > 0)
            {
              umts_deq_turn_ = 3;
         } else {
                    if (con_q_->length() > 0)
                       {
                         umts_deq_turn_ = 1;
                    } else {
                              if (str_q_->length() > 0)
                                 {
                                   umts_deq_turn_ = 2;
                              } else {
                                        if (bac_q_->length() > 0)
                                           {
                                             umts_deq_turn_ = 4;
                                        } else {
                                                  if (ovb_q_->length() > 0)
                                                     {
                                                       umts_deq_turn_ = 5;
                                                  }
                                           }
                                 }
                    }
            }
         break;
  }
  case 10:
        {
if (int_q_->length() > 0)
            {
              umts_deq_turn_ = 3;
         } else {
                    if (con_q_->length() > 0)
```

```
                {
                  umts_deq_turn_ = 1;
                } else {
                        if (str_q_->length() > 0)
                          {
                            umts_deq_turn_ = 2;
                        } else {
                                if (bac_q_->length() > 0)
                                  {
                                    umts_deq_turn_ = 4;
                                } else {
                                        if (ovb_q_->length() > 0)
                                          {
                                            umts_deq_turn_ = 5;
                                        }
                                  }
                          }
                }
        }
        break;
  }

  // to set dequeue turn to interactive traffic first if bac_q_->length()      //case (5):
  case 5:
        {
if (bac_q_->length() > 0)
        {
          umts_deq_turn_ = 4;
        } else {
                if (con_q_->length() > 0)
                  {
                    umts_deq_turn_ = 1;
                } else {
                        if (str_q_->length() > 0)
                          {
                            umts_deq_turn_ = 2;
                        } else {
                                if (int_q_->length() > 0)
                                  {
                                    umts_deq_turn_ = 3;
                                } else {
                                        if (ovb_q_->length() > 0)
                                          {
                                            umts_deq_turn_ = 5;
                                        }
                                  }
                          }
                }
        }
        break;
  }

  // to set dequeue turn to overflow buffer traffic first if ovb_q_->length()      //case (7, 11, 14):
  case 7:
        {
if (ovb_q_->length() > 0)
        {
          umts_deq_turn_ =5;
        } else {
                if (con_q_->length() > 0)
                  {
                    umts_deq_turn_ = 1;
                } else {
                        if (str_q_->length() > 0)
                          {
                            umts_deq_turn_ = 2;
```

```
                        } else {
                                if (int_q_->length() > 0)
                                   {
                                     umts_deq_turn_ = 3;
                                } else {
                                        if (bac_q_->length() > 0)
                                           {
                                             umts_deq_turn_ =4;
                                           }
                                   }
                             }
                        }
                  }
                break;
   }
   case 11:
         {
if (ovb_q_->length() > 0)
              {
                umts_deq_turn_ =5;
          } else {
                  if (con_q_->length() > 0)
                     {
                       umts_deq_turn_ = 1;
                     } else {
                             if (str_q_->length() > 0)
                                {
                                  umts_deq_turn_ = 2;
                             } else {
                                     if (int_q_->length() > 0)
                                        {
                                          umts_deq_turn_ = 3;
                                     } else {
                                             if (bac_q_->length() > 0)
                                                {
                                                  umts_deq_turn_ =4;
                                                }
                                        }
                                }
                        }
                  }
                break;
   }
   case 14:
         {
if (ovb_q_->length() > 0)
              {
                umts_deq_turn_ =5;
          } else {
                  if (con_q_->length() > 0)
                     {
                       umts_deq_turn_ = 1;
                     } else {
                             if (str_q_->length() > 0)
                                {
                                  umts_deq_turn_ = 2;
                             } else {
                                     if (int_q_->length() > 0)
                                        {
                                          umts_deq_turn_ = 3;
                                     } else {
                                             if (bac_q_->length() > 0)
                                                {
                                                  umts_deq_turn_ =4;
                                                }
                                        }
                                }
                        }
                  }
```

```
                                        }
                                    }
                                }
                            }
                            break;
    }
    default:
            {
             umts_deq_turn_ = 0;
             break;
    }
}

switch(umts_deq_turn_)
{
    case 1:
            {
             p = con_q_->deque();
             deque_packet = true;
            con_deq_ctr_++;
             output_packet_forwarding_delay(CON);
             break;
    }
    case 2:
            {
             p = str_q_->deque();
             deque_packet = true;
            str_deq_ctr_++;
             output_packet_forwarding_delay(STR);
             break;
    }
    case 3:
            {
             p = int_q_->deque();
             deque_packet = true;
            int_deq_ctr_++;
             output_packet_forwarding_delay(INT);
             break;
    }
    case 4:
            {
             p = bac_q_->deque();
             deque_packet = true;
            bac_deq_ctr_++;
             output_packet_forwarding_delay(BAC);
             break;
    }
    case 5:
            {
             p = ovb_q_->deque();
             deque_packet = true;
             hdr_ip* iph = hdr_ip::access(p);
             switch (iph->prio_)
             {
               case 15:
                     {
                          con_deq_ctr_++;
                          ovb_deq_ctr_++;
                          output_packet_forwarding_delay(CON);
                      break;
               }
               case 13:
                     {
                          str_deq_ctr_++;
                          ovb_deq_ctr_++;
```

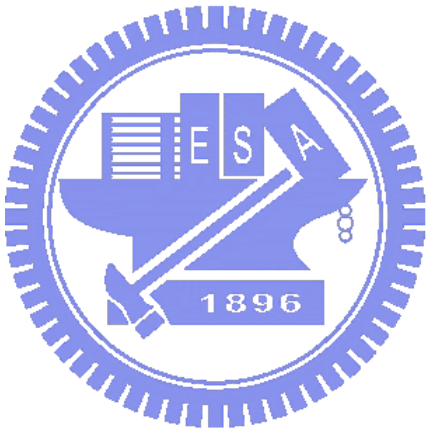
```
                    output_packet_forwarding_delay(STR);
                break;
            }
            case 11:
                {
                    int_deq_ctr_++;
                    ovb_deq_ctr_++;
                    output_packet_forwarding_delay(INT);
                break;
            }
            case 9:
                {
                    bac_deq_ctr_++;
                    ovb_deq_ctr_++;
                    output_packet_forwarding_delay(BAC);
                break;
            }
            default:
                {
                   printf("No UMTS packet avaialble dequeue from the overflow buffer !!!\n");
                break;
            }
           }
           break;
    }
    default:
          {
           break;
    }
   }
   if (deque_packet == true)
     {
      packet_deque_ctr_++;
      return (p);
   } else
         { return 0;}
}

// To record UMTS packets forwarding delay time
void UmtsPdbQueue::output_packet_forwarding_delay(int packet_type_)
{
 FILE *pkt_forwarding_delay_rec_fp;
 double cur_clock, cur_pkt_forwarding_delay_time;

 cur_clock = 0.0;
 switch (packet_type_)
     {
      case CON:
            {
             pkt_forwarding_delay_rec_fp=fopen("con_pkt_forwarding_delay_rec_pdb","a+");
             cur_clock = Scheduler::instance().clock();
             cur_pkt_forwarding_delay_time = cur_clock - prev_con_pkt_forwarding_clock;
             fprintf(pkt_forwarding_delay_rec_fp,"%f\n",cur_pkt_forwarding_delay_time);
             prev_con_pkt_forwarding_clock = cur_clock;
             break;
      }
      case STR:
            {
             pkt_forwarding_delay_rec_fp=fopen("str_pkt_forwarding_delay_rec_pdb","a+");
             cur_clock = Scheduler::instance().clock();
             cur_pkt_forwarding_delay_time = cur_clock - prev_str_pkt_forwarding_clock;
             fprintf(pkt_forwarding_delay_rec_fp,"%f\n",cur_pkt_forwarding_delay_time);
             prev_str_pkt_forwarding_clock = cur_clock;
             break;
      }
```

```
        case INT:
              {
               pkt_forwarding_delay_rec_fp=fopen("int_pkt_forwarding_delay_rec_pdb","a+");
               cur_clock = Scheduler::instance().clock();
               cur_pkt_forwarding_delay_time = cur_clock - prev_int_pkt_forwarding_clock;
               fprintf(pkt_forwarding_delay_rec_fp,"%f\n",cur_pkt_forwarding_delay_time);
               prev_int_pkt_forwarding_clock = cur_clock;
               break;
        }
        case BAC:
              {
               pkt_forwarding_delay_rec_fp=fopen("bac_pkt_forwarding_delay_rec_pdb","a+");
               cur_clock = Scheduler::instance().clock();
               cur_pkt_forwarding_delay_time = cur_clock - prev_bac_pkt_forwarding_clock;
               fprintf(pkt_forwarding_delay_rec_fp,"%f\n",cur_pkt_forwarding_delay_time);
               prev_bac_pkt_forwarding_clock = cur_clock;
               break;
        }
     default:
          {
           break;
     }
  }
  fclose(pkt_forwarding_delay_rec_fp);
}

// To print out the statistics information of UMTS priority queueing discipling// Unit : packet
void UmtsPdbQueue::printstats()
{
  printf("\n***    UMTS priority queueing discipline operation statistics    ***\n");
  printf("=============================================================\n");
  printf("Traffic type          enqueue          dequeue          dropped\n");
  printf("------------          -------          -------          -------\n");
  printf("Conversation          %7d              %7d %7d\n",con_enq_ctr_,con_deq_ctr_,con_dpq_ctr_);
  printf("Stream                %7d              %7d
   %7d\n",str_enq_ctr_,str_deq_ctr_,str_dpq_ctr_);
  printf("Interactive           %7d              %7d
   %7d\n",int_enq_ctr_,int_deq_ctr_,int_dpq_ctr_);
  printf("Background            %7d              %7d
   %7d\n",bac_enq_ctr_,bac_deq_ctr_,bac_dpq_ctr_);
  printf("\n-------------------------------------------------------------\n");
  printf("Total UMTS dequeue packet number from overflow buffer : %d\n",ovb_deq_ctr_);
  printf("Total UMTS dequeue packet number : %d\n",packet_deque_ctr_);
  printf("Total UMTS enqueue overflow buffer packet number : %d\n",ovb_enq_ctr_);

}


// int command(int argc, const char*const* argv)
// Commands from the ns file are interpreted through this interface.
int UmtsPdbQueue::command(int argc, const char*const* argv) {
if (strcmp(argv[1], "printstats") == 0) {
      printstats();
      return (TCL_OK);
   }
  return(Queue::command(argc, argv));

}
```

Appendix D: The OQB Header File

```
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include "queue.h"
#include "address.h"

#define CON 1                  // Packet type : conversational
#define STR 2                  // Packet type : streaming
#define INT 3                  // Packet type : interactive
#define BAC 4                  // Packet type : background

class UmtsPdbQueue : public Queue {
  public:
    UmtsPdbQueue();
    int command(int argc, const char*const* argv);          // interface to ns scripts

  protected:
                void enque(Packet*);
                unsigned long random_seed_by_system_time();
                bool overflow_pkt_enqueue_by_prob(int enq_min, int enq_max, int enq_pkt_num, double
               dbenque_prob);
    Packet* deque();
                void output_packet_forwarding_delay(int packet_type_);
    void printstats();

PacketQueue *con_q_;        // the FIFO queue for UMTS conversation traffic
PacketQueue *str_q_;        // the FIFO queue for UMTS stream traffic
PacketQueue *int_q_;        // the FIFO queue for UMTS interactive traffic
PacketQueue *bac_q_;        // the FIFO queue for UMTS background traffic
PacketQueue *ovb_q_;        // the FIFO queue for UMTS overflow buffer
bool con_enable_enque_;    // boolean for UMTS conversational packet enqueue
bool str_enable_enque_;    // boolean for UMTS streaming packet enqueue
bool int_enable_enque_;    // boolean for UMTS interactive packet enqueue
bool bac_enable_enque_;    // boolean for UMTS background packet enqueue
bool ovb_enable_enque_;    // boolean for UMTS overflow buffer packet enqueue

int packet_deque_ctr_;    // to count the sent packets
int pkt_deque_cycle_;      // to mod the sent packets
int umts_deq_turn_;         // 1 for con_q, 2 for str_q, 3 for int_q, 4 for bac_q
double prev_con_pkt_forwarding_clock;    // Previous conversational packet forwarding clock
double prev_str_pkt_forwarding_clock;    // Previous streaming packet forwarding clock
double prev_int_pkt_forwarding_clock;    // Previous nteractive packet forwarding clock
double prev_bac_pkt_forwarding_clock;    // Previous background packet forwarding clock

    int phy_q_length_;        // used queue buffer length
    int con_enq_ctr_;        // UMTS conversation application enqueue packet counter
    int str_enq_ctr_;      // UMTS stream application enqueue packet counter
    int int_enq_ctr_;      // UMTS interactive application enqueue packet counter
    int bac_enq_ctr_;     // UMTS background application enqueue packet counter
    int ovb_enq_ctr_;     // UMTS overflow buffer enqueue packet counter
    int con_deq_ctr_;     // UMTS conversation application dequeue packet counter
    int str_deq_ctr_;      // UMTS stream application dequeue packet counter
    int int_deq_ctr_;      // UMTS interactive application dequeue packet counter
    int bac_deq_ctr_;     // UMTS background application dequeue packet counter
    int ovb_deq_ctr_;     // UMTS overflow buffer dequeue packet counter
    int con_dpq_ctr_;     // UMTS conversation application packet drop by queue
    int str_dpq_ctr_;      // UMTS stream application packet drop by queue
    int int_dpq_ctr_;      // UMTS interactive application packet drop by queue
    int bac_dpq_ctr_;     // UMTS background application packet drop by queue
    int con_queue_limit_;       // UMTS conversational application queue limit
    int str_queue_limit_;        // UMTS streaming application queue limit
    int int_queue_limit_;        // UMTS interactive application queue limit
```

```
 int bac_queue_limit_;       // UMTS background application queue limit
int c_enq_min_;              // UMTS conversational application min queue limit
 int c_enq_max_;       // UMTS conversation application max queue limit
 int s_enq_min_;       // UMTS streaming application min queue limit
 int s_enq_max_;       // UMTS streaming application max queue limit
 int i_enq_min_;       // UMTS interactive application min queue limit
 int i_enq_max_;       // UMTS interactive application max queue limit
 int b_enq_min_;       // UMTS background application min queue limit
 int b_enq_max_;       // UMTS background application max queue limit
double con_dbenque_prob_;          //UMTS conversational aplication packet enqueue probability
double str_dbenque_prob_;       //UMTS streaming aplication packet enqueue probability
double int_dbenque_prob_;       //UMTS interactive aplication packet enqueue probability
double bac_dbenque_prob_;       //UMTS background aplication packet enqueue probability
};
```



Appendix E: Simulation scenarios for DQB in a continuous traffic pattern

```
set ns [new Simulator]
#Define different colors for data flows
$ns color 1 Red
$ns color 2 Blue
$ns color 3 green
$ns color 4 yellow
$ns color 5 black
#Open the nam trace file
set tf [open umts-pred-out-a.tr w]
$ns trace-all $tf
#Define a 'finish' procedure
proc finish {}
{
   global ns tf
   $ns flush-trace
   close $tf
   #Execute trace file process
   set PERL "/usr/bin/perl"
   set USERHOME [exec env | grep "^HOME" | sed /^HOME=/s/^HOME=//]
   set NSHOME "$USERHOME/ns2/ns-allinone-2.30"
   set XGRAPH "$NSHOME/bin/xgraph"
   set GETDRT "$NSHOME/ns-2.30/bin/getdrt"
   exec $PERL $GETDRT -s 0.0 -d 6.0 -f 1 umts-pred-out-a.tr > predumtscdrta.tr
exec $PERL $GETDRT -s 1.0 -d 7.0 -f 2 umts-pred-out-a.tr > predumtssdrta.tr
exec $PERL $GETDRT -s 2.0 -d 8.0 -f 3 umts-pred-out-a.tr > predumtsidrta.tr
exec $PERL $GETDRT -s 3.0 -d 9.0 -f 4 umts-pred-out-a.tr > predumtsbdrta.tr
exit 0
}
set node_(s1) [$ns node]
set node_(s2) [$ns node]
set node_(s3) [$ns node]
set node_(s4) [$ns node]
set node_(g1) [$ns node]
set node_(g2) [$ns node]
set node_(r1) [$ns node]
set node_(r2) [$ns node]
set node_(r3) [$ns node]
set node_(r4) [$ns node]
$ns duplex-link $node_(s1) $node_(g1) 2.0Mb 1ms DropTail
$ns duplex-link $node_(s2) $node_(g1) 2.0Mb 1ms DropTail
$ns duplex-link $node_(s3) $node_(g1) 2.0Mb 1ms DropTail
$ns duplex-link $node_(s4) $node_(g1) 2.0Mb 1ms DropTail
$ns duplex-link $node_(g1) $node_(g2) 3.0Mb 100ms UmtsPredQueue
$ns duplex-link $node_(r1) $node_(g2) 2.0Mb 1ms DropTail
$ns duplex-link $node_(r2) $node_(g2) 2.0Mb 1ms DropTail
$ns duplex-link $node_(r3) $node_(g2) 2.0Mb 1ms DropTail
$ns duplex-link $node_(r4) $node_(g2) 2.0Mb 1ms DropTail
set umtspredqueue [[$ns link $node_(g1) $node_(g2)] queue]

#Setup UmtsPdbQueue queue parameter
$ns queue-limit $node_(g1) $node_(g2) 40
$umtspredqueue set c_enq_min 8
$umtspredqueue set s_enq_min 7
$umtspredqueue set i_enq_min 6
$umtspredqueue set b_enq_min 5
$umtspredqueue set c_enq_max 40
$umtspredqueue set s_enq_max 39
$umtspredqueue set i_enq_max 38
$umtspredqueue set b_enq_max 37
$umtspredqueue set con_dbenque_prob 1.0
```

```
$umtspredqueue set str_dbenque_prob 0.95
$umtspredqueue set int_dbenque_prob 0.90
$umtspredqueue set bac_dbenque_prob 0.85
$umtspredqueue set con_queue_limit 8
$umtspredqueue set str_queue_limit 7
$umtspredqueue set int_queue_limit 6
$umtspredqueue set bac_queue_limit 5

$ns duplex-link-op $node_(g1) $node_(g2) queuePos 0.5
$ns duplex-link-op $node_(s1) $node_(g1) orient up
$ns duplex-link-op $node_(s2) $node_(g1) orient left-up
$ns duplex-link-op $node_(s3) $node_(g1) orient left-down
$ns duplex-link-op $node_(s4) $node_(g1) orient down
$ns duplex-link-op $node_(g1) $node_(g2) orient right
$ns duplex-link-op $node_(g2) $node_(r1) orient up
$ns duplex-link-op $node_(g2) $node_(r2) orient rigth-up
$ns duplex-link-op $node_(g2) $node_(r3) orient rigth-down
$ns duplex-link-op $node_(g2) $node_(r4) orient down

#Setup a UMTS UDP connection
set udp_s1 [new Agent/UDP/UDPUmtsc]
set udp_r1 [new Agent/UDP/UDPUmtsc]
$ns attach-agent $node_(s1) $udp_s1
$ns attach-agent $node_(r1) $udp_r1
$ns connect $udp_s1 $udp_r1
set udp_con_pktsize 1000
$udp_s1 set packetSize_ $udp_con_pktsize
$udp_r1 set packetSize_ $udp_con_pktsize
$udp_s1 set fid_ 1
$udp_r1 set fid_ 1

set udp_s2 [new Agent/UDP/UDPUmtss]
set udp_r2 [new Agent/UDP/UDPUmtss]
$ns attach-agent $node_(s2) $udp_s2
$ns attach-agent $node_(r2) $udp_r2
$ns connect $udp_s2 $udp_r2
set udp_str_pktsize 1000
$udp_s2 set packetSize_ $udp_str_pktsize
$udp_r2 set packetSize_ $udp_str_pktsize
$udp_s2 set fid_ 2
$udp_r2 set fid_ 2

set udp_s3 [new Agent/UDP/UDPUmtsi]
set udp_r3 [new Agent/UDP/UDPUmtsi]
$ns attach-agent $node_(s3) $udp_s3
$ns attach-agent $node_(r3) $udp_r3
$ns connect $udp_s3 $udp_r3
set udp_int_pktsize 1000
$udp_s3 set packetSize_ $udp_int_pktsize
$udp_r3 set packetSize_ $udp_int_pktsize
$udp_s3 set fid_ 3
$udp_r3 set fid_ 3

set udp_s4 [new Agent/UDP/UDPUmtsb]
set udp_r4 [new Agent/UDP/UDPUmtsb]
$ns attach-agent $node_(s4) $udp_s4
$ns attach-agent $node_(r4) $udp_r4
$ns connect $udp_s4 $udp_r4
set udp_bac_pktsize 1000
$udp_s4 set packetSize_ $udp_bac_pktsize
$udp_r4 set packetSize_ $udp_bac_pktsize
$udp_s4 set fid_ 4
$udp_r4 set fid_ 4

#Setup a UMTS Conversation Application
```

```
set umtscapp_s [new Application/UMTSCApp]
set umtscapp_r [new Application/UMTSCApp]
$umtscapp_s attach-agent $udp_s1
$umtscapp_r attach-agent $udp_r1
$umtscapp_s set pktsize_ 1000
$umtscapp_s set random_ false

#Setup a UMTS stream Application
set umtssapp_s [new Application/UMTSSApp]
set umtssapp_r [new Application/UMTSSApp]
$umtssapp_s attach-agent $udp_s2
$umtssapp_r attach-agent $udp_r2
$umtssapp_s set pktsize_ 1000
$umtssapp_s set random_ false

#Setup a UMTS Interactive Application
set umtsiapp_s [new Application/UMTSIApp]
set umtsiapp_r [new Application/UMTSIApp]
$umtsiapp_s attach-agent $udp_s3
$umtsiapp_r attach-agent $udp_r3
$umtsiapp_s set pktsize_ 1000
$umtsiapp_s set random_ false

#Setup a UMTS Background Application
set umtsbapp_s [new Application/UMTSBApp]
set umtsbapp_r [new Application/UMTSBApp]
$umtsbapp_s attach-agent $udp_s4
$umtsbapp_r attach-agent $udp_r4
$umtsbapp_s set pktsize_ 1000
$umtsbapp_s set random_ false

set start_all_time 0.0
set stop_all_time 30.0

$ns at $start_all_time "$umtscapp_s start"
$ns at $start_all_time "$umtssapp_s start"
$ns at $start_all_time "$umtsiapp_s start"
$ns at $start_all_time "$umtsbapp_s start"
$ns at $stop_all_time "$umtscapp_s stop"
$ns at $stop_all_time "$umtssapp_s stop"
$ns at $stop_all_time "$umtsiapp_s stop"
$ns at $stop_all_time "$umtsbapp_s stop"
$ns at 31.0 "$umtspredqueue printstats"
$ns at 31.0 "finish"

$ns run
```

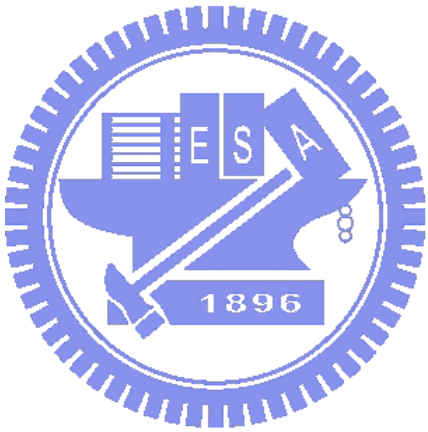
Appendix F: Simulation scenarios for OQB in a continuous traffic pattern

```
set ns [new Simulator]
#Define different colors for data flows
$ns color 1 Red
$ns color 2 Blue
$ns color 3 green
$ns color 4 yellow
$ns color 5 black

#Open the nam trace file
set tf [open umts-pdb-out-a.tr w]
$ns trace-all $tf
#Define a 'finish' procedure
proc finish {}
{
        global ns tf
        $ns flush-trace
        close $tf
         #Execute trace file process
           set PERL "/usr/bin/perl"
            set USERHOME [exec env | grep "^HOME" | sed /^HOME=/s/^HOME=//]
           set NSHOME "$USERHOME/ns2/ns-allinone-2.30"
           set XGRAPH "$NSHOME/bin/xgraph"
           set GETDRT "$NSHOME/ns-2.30/bin/getdrt"
            exec $PERL $GETDRT -s 0.0 -d 6.0 -f 1 umts-pdb-out-a.tr > predumtscdrta.tr
            exec $PERL $GETDRT -s 1.0 -d 7.0 -f 2 umts-pdb-out-a.tr > predumtssdrta.tr
            exec $PERL $GETDRT -s 2.0 -d 8.0 -f 3 umts-pdb-out-a.tr > predumtsidrta.tr
            exec $PERL $GETDRT -s 3.0 -d 9.0 -f 4 umts-pdb-out-a.tr > predumtsbdrta.tr
           exit 0
}
set node_(s1) [$ns node]
set node_(s2) [$ns node]
set node_(s3) [$ns node]
set node_(s4) [$ns node]
set node_(g1) [$ns node]
set node_(g2) [$ns node]
set node_(r1) [$ns node]
set node_(r2) [$ns node]
set node_(r3) [$ns node]
set node_(r4) [$ns node]

$ns duplex-link $node_(s1) $node_(g1) 2.0Mb 1ms DropTail
$ns duplex-link $node_(s2) $node_(g1) 2.0Mb 1ms DropTail
$ns duplex-link $node_(s3) $node_(g1) 2.0Mb 1ms DropTail
$ns duplex-link $node_(s4) $node_(g1) 2.0Mb 1ms DropTail
$ns duplex-link $node_(g1) $node_(g2) 3.0Mb 100ms UmtsPdbQueue
$ns duplex-link $node_(r1) $node_(g2) 2.0Mb 1ms DropTail
$ns duplex-link $node_(r2) $node_(g2) 2.0Mb 1ms DropTail
$ns duplex-link $node_(r3) $node_(g2) 2.0Mb 1ms DropTail
$ns duplex-link $node_(r4) $node_(g2) 2.0Mb 1ms DropTail
set umtspdbqueue [[$ns link $node_(g1) $node_(g2)] queue]
#Setup UmtsPdbQueue queue parameter
$ns queue-limit $node_(g1) $node_(g2) 40
```

```
$umtspdbqueue set c_enq_min 8
$umtspdbqueue set s_enq_min 7
$umtspdbqueue set i_enq_min 6
$umtspdbqueue set b_enq_min 5
$umtspdbqueue set c_enq_max 40
$umtspdbqueue set s_enq_max 38
$umtspdbqueue set i_enq_max 36
$umtspdbqueue set b_enq_max 34
$umtspdbqueue set con_dbenque_prob 1.0
$umtspdbqueue set str_dbenque_prob 0.9
$umtspdbqueue set int_dbenque_prob 0.8
$umtspdbqueue set bac_dbenque_prob 0.7
$umtspdbqueue set con_queue_limit 8
$umtspdbqueue set str_queue_limit 7
$umtspdbqueue set int_queue_limit 6
$umtspdbqueue set bac_queue_limit 5

$ns duplex-link-op $node_(g1) $node_(g2) queuePos 0.5
$ns duplex-link-op $node_(s1) $node_(g1) orient up
$ns duplex-link-op $node_(s2) $node_(g1) orient left-up
$ns duplex-link-op $node_(s3) $node_(g1) orient left-down
$ns duplex-link-op $node_(s4) $node_(g1) orient down
$ns duplex-link-op $node_(g1) $node_(g2) orient right
$ns duplex-link-op $node_(g2) $node_(r1) orient up
$ns duplex-link-op $node_(g2) $node_(r2) orient rigth-up
$ns duplex-link-op $node_(g2) $node_(r3) orient rigth-down
$ns duplex-link-op $node_(g2) $node_(r4) orient down

#Setup a UMTS UDP connection
set udp_s1 [new Agent/UDP/UDPUmtsc]
set udp_r1 [new Agent/UDP/UDPUmtsc]
$ns attach-agent $node_(s1) $udp_s1
$ns attach-agent $node_(r1) $udp_r1
$ns connect $udp_s1 $udp_r1
set udp_con_pktsize 1000
$udp_s1 set packetSize_ $udp_con_pktsize
$udp_r1 set packetSize_ $udp_con_pktsize
$udp_s1 set fid_ 1
$udp_r1 set fid_ 1

set udp_s2 [new Agent/UDP/UDPUmtss]
set udp_r2 [new Agent/UDP/UDPUmtss]
$ns attach-agent $node_(s2) $udp_s2
$ns attach-agent $node_(r2) $udp_r2
$ns connect $udp_s2 $udp_r2
set udp_str_pktsize 1000
$udp_s2 set packetSize_ $udp_str_pktsize
$udp_r2 set packetSize_ $udp_str_pktsize
$udp_s2 set fid_ 2
$udp_r2 set fid_ 2

set udp_s3 [new Agent/UDP/UDPUmtsi]
set udp_r3 [new Agent/UDP/UDPUmtsi]
$ns attach-agent $node_(s3) $udp_s3
$ns attach-agent $node_(r3) $udp_r3
$ns connect $udp_s3 $udp_r3
```

```
set udp_int_pktsize 1000
$udp_s3 set packetSize_ $udp_int_pktsize
$udp_r3 set packetSize_ $udp_int_pktsize
$udp_s3 set fid_ 3
$udp_r3 set fid_ 3

set udp_s4 [new Agent/UDP/UDPUmtsb]
set udp_r4 [new Agent/UDP/UDPUmtsb]
$ns attach-agent $node_(s4) $udp_s4
$ns attach-agent $node_(r4) $udp_r4
$ns connect $udp_s4 $udp_r4
set udp_bac_pktsize 1000
$udp_s4 set packetSize_ $udp_bac_pktsize
$udp_r4 set packetSize_ $udp_bac_pktsize
$udp_s4 set fid_ 4
$udp_r4 set fid_ 4
#Setup a UMTS Conversation Application
set umtscapp_s [new Application/UMTSCApp]
set umtscapp_r [new Application/UMTSCApp]
$umtscapp_s attach-agent $udp_s1
$umtscapp_r attach-agent $udp_r1
$umtscapp_s set pktsize_ 1000
$umtscapp_s set random_ false

#Setup a UMTS stream Application
set umtssapp_s [new Application/UMTSSApp]
set umtssapp_r [new Application/UMTSSApp]
$umtssapp_s attach-agent $udp_s2
$umtssapp_r attach-agent $udp_r2
$umtssapp_s set pktsize_ 1000
$umtssapp_s set random_ false

#Setup a UMTS Interactive Application
set umtsiapp_s [new Application/UMTSIApp]
set umtsiapp_r [new Application/UMTSIApp]
$umtsiapp_s attach-agent $udp_s3
$umtsiapp_r attach-agent $udp_r3
$umtsiapp_s set pktsize_ 1000
$umtsiapp_s set random_ false

#Setup a UMTS Background Application
set umtsbapp_s [new Application/UMTSBApp]
set umtsbapp_r [new Application/UMTSBApp]
$umtsbapp_s attach-agent $udp_s4
$umtsbapp_r attach-agent $udp_r4
$umtsbapp_s set pktsize_ 1000
$umtsbapp_s set random_ false

set start_all_time 0.0
set stop_all_time 30.0

$ns at $start_all_time "$umtscapp_s start"
$ns at $start_all_time "$umtssapp_s start"
$ns at $start_all_time "$umtsiapp_s start"
$ns at $start_all_time "$umtsbapp_s start"
$ns at $stop_all_time "$umtscapp_s stop"
```

```
$ns at $stop_all_time "$umtssapp_s stop"
$ns at $stop_all_time "$umtsiapp_s stop"
$ns at $stop_all_time "$umtsbapp_s stop"
$ns at 31.0 "$umtspdbqueue printstats"
$ns at 31.0 "finish"

$ns run
```



Appendix G: Simulation scenarios for DQB in a intermittent traffic pattern

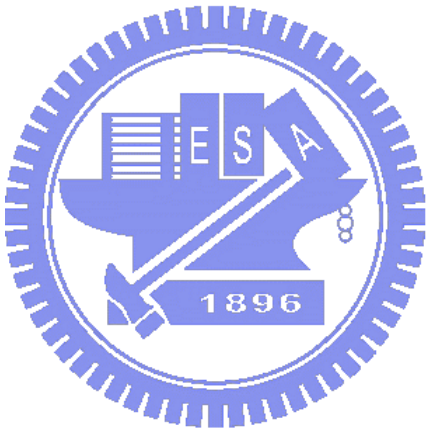
```
set ns [new Simulator]
#Define different colors for data flows
$ns color 1 Red
$ns color 2 Blue
$ns color 3 green
$ns color 4 yellow
$ns color 5 black

#Open the nam trace file
set tf [open umts-pred-out-p.tr w]
$ns trace-all $tf

#Define a 'output' pricedure
proc output {} {
  global udp_r1 udp_r2 udp_r3 udp_r4
  global con_xmit_period str_xmit_period int_xmit_period bac_xmit_period
  global udp_con_pktsize udp_str_pktsize udp_int_pktsize udp_bac_pktsize
  set con_app_rbytes [$udp_r1 set con_rbytes]
  set str_app_rbytes [$udp_r2 set str_rbytes]
  set int_app_rbytes [$udp_r3 set int_rbytes]
  set bac_app_rbytes [$udp_r4 set bac_rbytes]
  puts " "
  puts "con_xmit_period : $con_xmit_period"
  puts "str_xmit_period : $str_xmit_period"
  puts "int_xmit_period : $int_xmit_period"
  puts "bac_xmit_period : $bac_xmit_period"
  puts "Conversation application transmission volume : $con_app_rbytes"
  puts "Stream application transmission volume : $str_app_rbytes"
  puts "Interactive application transmission volume : $int_app_rbytes"
  puts "Background application transmission volume : $bac_app_rbytes"
  set con_app_xmit_performance_byte [expr $con_app_rbytes / $con_xmit_period]
  set str_app_xmit_performance_byte [expr $str_app_rbytes / $str_xmit_period]
  set int_app_xmit_performance_byte [expr $int_app_rbytes / $int_xmit_period]
  set bac_app_xmit_performance_byte [expr $bac_app_rbytes / $bac_xmit_period]
  puts "    "
  puts "Conversation application transmission performance by bytes : $con_app_xmit_performance_byte"
  puts "Stream application transmission performance by bytes : $str_app_xmit_performance_byte"
  puts "Interactive application transmission performance by bytes : $int_app_xmit_performance_byte"
  puts "Background application transmission performance by bytes : $bac_app_xmit_performance_byte"
  set con_app_xmit_performance_pkt [expr [expr $con_app_rbytes / $udp_con_pktsize] /
        $con_xmit_period]
  set str_app_xmit_performance_pkt [expr [expr $str_app_rbytes / $udp_str_pktsize] / $str_xmit_period]
  set int_app_xmit_performance_pkt [expr [expr $int_app_rbytes / $udp_int_pktsize] / $int_xmit_period]
  set bac_app_xmit_performance_pkt [expr [expr $bac_app_rbytes / $udp_bac_pktsize] / $bac_xmit_period]
  puts "    "
  puts "Conversation application transmission performance by packets : $con_app_xmit_performance_pkt"
  puts "Stream application transmission performance by packets : $str_app_xmit_performance_pkt"
  puts "Interactive application transmission performance by packets : $int_app_xmit_performance_pkt"
  puts "Background application transmission performance by packets : $bac_app_xmit_performance_pkt"
}

#Define a 'finish' procedure
proc finish {} {
        global ns tf
        $ns flush-trace
        close $tf

#Execute trace file process
set PERL "/usr/bin/perl"
set USERHOME [exec env | grep "^HOME" | sed /^HOME=/s/^HOME=//]
```

```
set NSHOME "$USERHOME/ns2/ns-allinone-2.30"
set XGRAPH "$NSHOME/bin/xgraph"
set GETSET "$NSHOME/ns-2.30/bin/getset"
set GETDRT "$NSHOME/ns-2.30/bin/getdrt"
set EEDELAYS "$NSHOME/ns-2.30/bin/eedelay_s"
exec $PERL $GETDRT -s 0.0 -d 6.0 -f 1 umts-pred-out-p.tr > predumtscdrtp.tr
exec $PERL $GETDRT -s 1.0 -d 7.0 -f 2 umts-pred-out-p.tr > predumtssdrtp.tr
exec $PERL $GETDRT -s 2.0 -d 8.0 -f 3 umts-pred-out-p.tr > predumtsidrtp.tr
exec $PERL $GETDRT -s 3.0 -d 9.0 -f 4 umts-pred-out-p.tr > predumtsbdrtp.tr
        exit 0
}
set node_(s1) [$ns node]
set node_(s2) [$ns node]
set node_(s3) [$ns node]
set node_(s4) [$ns node]
set node_(g1) [$ns node]
set node_(g2) [$ns node]
set node_(r1) [$ns node]
set node_(r2) [$ns node]
set node_(r3) [$ns node]
set node_(r4) [$ns node]

$ns duplex-link $node_(s1) $node_(g1) 2.0Mb 1ms DropTail
$ns duplex-link $node_(s2) $node_(g1) 2.0Mb 1ms DropTail
$ns duplex-link $node_(s3) $node_(g1) 2.0Mb 1ms DropTail
$ns duplex-link $node_(s4) $node_(g1) 2.0Mb 1ms DropTail
$ns duplex-link $node_(g1) $node_(g2) 3Mb 100ms UmtsPredQueue
$ns duplex-link $node_(r1) $node_(g2) 2.0Mb 1ms DropTail
$ns duplex-link $node_(r2) $node_(g2) 2.0Mb 1ms DropTail
$ns duplex-link $node_(r3) $node_(g2) 2.0Mb 1ms DropTail
$ns duplex-link $node_(r4) $node_(g2) 2.0Mb 1ms DropTail
set umtspredqueue [[$ns link $node_(g1) $node_(g2)] queue]

#Setup UmtsPdbQueue queue parameter
$ns queue-limit $node_(g1) $node_(g2) 40
$umtspredqueue set c_enq_min 8
$umtspredqueue set s_enq_min 7
$umtspredqueue set i_enq_min 6
$umtspredqueue set b_enq_min 5
$umtspredqueue set c_enq_max 40
$umtspredqueue set s_enq_max 38
$umtspredqueue set i_enq_max 36
$umtspredqueue set b_enq_max 34
$umtspredqueue set con_redenque_prob 1.0
$umtspredqueue set str_redenque_prob 0.9
$umtspredqueue set int_redenque_prob 0.8
$umtspredqueue set bac_redenque_prob 0.7
$umtspredqueue set con_queue_limit 8
$umtspredqueue set str_queue_limit 7
$umtspredqueue set int_queue_limit 6
$umtspredqueue set bac_queue_limit 5

$ns duplex-link-op $node_(g1) $node_(g2) queuePos 0.5
$ns duplex-link-op $node_(s1) $node_(g1) orient up
$ns duplex-link-op $node_(s2) $node_(g1) orient left-up
$ns duplex-link-op $node_(s3) $node_(g1) orient left-down
$ns duplex-link-op $node_(s4) $node_(g1) orient down
$ns duplex-link-op $node_(g1) $node_(g2) orient right
$ns duplex-link-op $node_(g2) $node_(r1) orient up
$ns duplex-link-op $node_(g2) $node_(r2) orient rigth-up
$ns duplex-link-op $node_(g2) $node_(r3) orient rigth-down
$ns duplex-link-op $node_(g2) $node_(r4) orient down
#Setup a UMTS UDP connection
set udp_s1 [new Agent/UDP/UDPUmtsc]
set udp_r1 [new Agent/UDP/UDPUmtsc]
```

```
$ns attach-agent $node_(s1) $udp_s1
$ns attach-agent $node_(r1) $udp_r1
$ns connect $udp_s1 $udp_r1
set udp_con_pktsize 1000
$udp_s1 set packetSize_ $udp_con_pktsize
$udp_r1 set packetSize_ $udp_con_pktsize
$udp_s1 set fid_ 1
$udp_r1 set fid_ 1

set udp_s2 [new Agent/UDP/UDPUmtss]
set udp_r2 [new Agent/UDP/UDPUmtss]
$ns attach-agent $node_(s2) $udp_s2
$ns attach-agent $node_(r2) $udp_r2
$ns connect $udp_s2 $udp_r2
set udp_str_pktsize 1000
$udp_s2 set packetSize_ $udp_str_pktsize
$udp_r2 set packetSize_ $udp_str_pktsize
$udp_s2 set fid_ 2
$udp_r2 set fid_ 2

set udp_s3 [new Agent/UDP/UDPUmtsi]
set udp_r3 [new Agent/UDP/UDPUmtsi]
$ns attach-agent $node_(s3) $udp_s3
$ns attach-agent $node_(r3) $udp_r3
$ns connect $udp_s3 $udp_r3
set udp_int_pktsize 1000
$udp_s3 set packetSize_ $udp_int_pktsize
$udp_r3 set packetSize_ $udp_int_pktsize
$udp_s3 set fid_ 3
$udp_r3 set fid_ 3
set udp_s4 [new Agent/UDP/UDPUmtsb]
set udp_r4 [new Agent/UDP/UDPUmtsb]
$ns attach-agent $node_(s4) $udp_s4
$ns attach-agent $node_(r4) $udp_r4
$ns connect $udp_s4 $udp_r4
set udp_bac_pktsize 1000
$udp_s4 set packetSize_ $udp_bac_pktsize
$udp_r4 set packetSize_ $udp_bac_pktsize
$udp_s4 set fid_ 4
$udp_r4 set fid_ 4

#Setup a UMTS Conversation Application
set umtscapp_s [new Application/UMTSCApp]
set umtscapp_r [new Application/UMTSCApp]
$umtscapp_s attach-agent $udp_s1
$umtscapp_r attach-agent $udp_r1
$umtscapp_s set pktsize_ 1000
$umtscapp_s set random_ false

#Setup a UMTS stream Application
set umtssapp_s [new Application/UMTSSApp]
set umtssapp_r [new Application/UMTSSApp]
$umtssapp_s attach-agent $udp_s2
$umtssapp_r attach-agent $udp_r2
$umtssapp_s set pktsize_ 1000
$umtssapp_s set random_ false

#Setup a UMTS Interactive Application
set umtsiapp_s [new Application/UMTSIApp]
set umtsiapp_r [new Application/UMTSIApp]
$umtsiapp_s attach-agent $udp_s3
$umtsiapp_r attach-agent $udp_r3
$umtsiapp_s set pktsize_ 1000
$umtsiapp_s set random_ false
```

```
#Setup a UMTS Background Application
set umtsbapp_s [new Application/UMTSBApp]
set umtsbapp_r [new Application/UMTSBApp]
$umtsbapp_s attach-agent $udp_s4
$umtsbapp_r attach-agent $udp_r4
$umtsbapp_s set pktsize_ 1000
$umtsbapp_s set random_ false

#Simulation Scenario
set start_all_time 0.0
set stop_con_time1 7.5
set stop_str_time1 9.5
set stop_int_time1 10.5
set start_con_time1 10.25
set start_str_time1 12.0
set start_int_time1 13.0
set stop_str_time2 14.25
set stop_int_time2 15.75
set stop_con_time2 16.0
set start_int_time2 17.25
set start_con_time2 17.6
set start_str_time2 18.0
set stop_con_time3 20.0
set stop_str_time3 20.6
set stop_int_time3 21.35
set start_str_time3 24.4
set start_int_time3 25.5
set start_con_time3 27.0
set stop_all_time 30.0
set con_xmit_period 0.0
set str_xmit_period 0.0
set int_xmit_period 0.0
set bac_xmit_period 0.0
set con_xmit_period [expr [expr $stop_con_time1 - $start_all_time] + [expr $stop_con_time2 -
$start_con_time1 ] + [expr $stop_con_time3 - $start_con_time2 ] + [expr $stop_all_time -
$start_con_time3]]
 set str_xmit_period [expr [expr $stop_str_time1 - $start_all_time] + [expr $stop_str_time2 -
 $start_str_time1 ] + [expr $stop_str_time3 - $start_str_time2 ] + [expr $stop_all_time - $start_str_time3]]
 set int_xmit_period [expr [expr $stop_int_time1 - $start_all_time] + [expr $stop_int_time2 -
 $start_int_time1 ] + [expr $stop_int_time3 - $start_int_time2 ] + [expr $stop_all_time - $start_int_time3]]
set bac_xmit_period $stop_all_time

$ns at $start_all_time "$umtscapp_s start"
$ns at $start_all_time "$umtssapp_s start"
$ns at $start_all_time "$umtsiapp_s start"
$ns at $start_all_time "$umtsbapp_s start"
$ns at $stop_con_time1 "$umtscapp_s stop"
$ns at $stop_str_time1 "$umtssapp_s stop"
$ns at $start_con_time1 "$umtscapp_s start"
$ns at $stop_int_time1 "$umtsiapp_s stop"
$ns at $start_str_time1 "$umtssapp_s start"
$ns at $start_int_time1 "$umtsiapp_s start"
$ns at $stop_str_time2 "$umtssapp_s stop"
$ns at $stop_int_time2 "$umtsiapp_s stop"
$ns at $stop_con_time2 "$umtscapp_s stop"
$ns at $start_int_time2 "$umtsiapp_s start"
$ns at $start_con_time2 "$umtscapp_s start"
$ns at $start_str_time2 "$umtssapp_s start"
$ns at $stop_con_time3 "$umtscapp_s stop"
$ns at $stop_str_time3 "$umtssapp_s stop"
$ns at $stop_int_time3 "$umtsiapp_s stop"
$ns at $start_str_time3 "$umtssapp_s start"
$ns at $start_int_time3 "$umtsiapp_s start"
$ns at $start_con_time3 "$umtscapp_s start"
$ns at $stop_all_time "$umtscapp_s stop"
```

```
$ns at $stop_all_time "$umtssapp_s stop"
$ns at $stop_all_time "$umtsiapp_s stop"
$ns at $stop_all_time "$umtsbapp_s stop"
$ns at 31.0 "$umtspredqueue printstats"
$ns at 31.0 "output"
$ns at 31.0 "finish"
$ns run
```

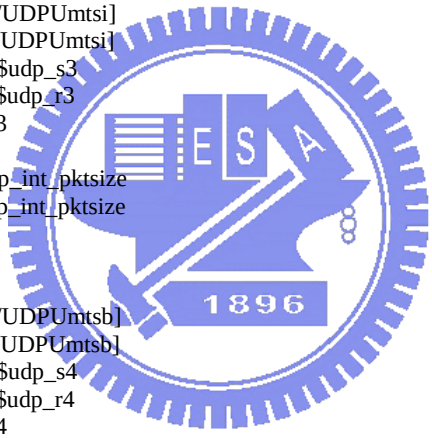


Appendix H: Simulation scenarios for OQB in a intermittent traffic pattern

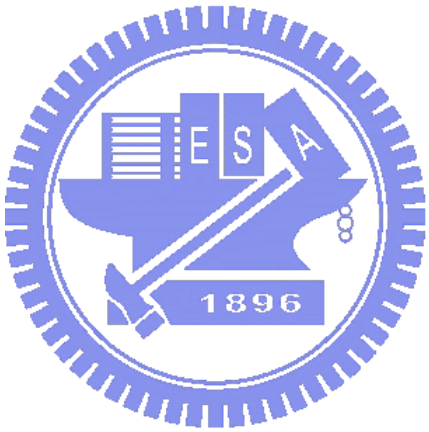
```
set ns [new Simulator]

#Define different colors for data flows
$ns color 1 Red
$ns color 2 Blue
$ns color 3 green
$ns color 4 yellow
$ns color 5 black

#Open the nam trace file
set tf [open umts-pdb-out-p.tr w]
$ns trace-all $tf

#Define a 'output' pricedure
proc output {} {
  global udp_r1 udp_r2 udp_r3 udp_r4
  global con_xmit_period str_xmit_period int_xmit_period bac_xmit_period
  global udp_con_pktsize udp_str_pktsize udp_int_pktsize udp_bac_pktsize

  set con_app_rbytes [$udp_r1 set con_rbytes]
  set str_app_rbytes [$udp_r2 set str_rbytes]
  set int_app_rbytes [$udp_r3 set int_rbytes]
  set bac_app_rbytes [$udp_r4 set bac_rbytes]

puts " "
puts "con_xmit_period : $con_xmit_period"
puts "str_xmit_period : $str_xmit_period"
puts "int_xmit_period : $int_xmit_period"
puts "bac_xmit_period : $bac_xmit_period"
puts "Conversation application transmission volume : $con_app_rbytes"
puts "Stream application transmission volume : $str_app_rbytes"
puts "Interactive application transmission volume : $int_app_rbytes"
puts "Background application transmission volume : $bac_app_rbytes"
set con_app_xmit_performance_byte [expr $con_app_rbytes / $con_xmit_period]
set str_app_xmit_performance_byte [expr $str_app_rbytes / $str_xmit_period]
set int_app_xmit_performance_byte [expr $int_app_rbytes / $int_xmit_period]
set bac_app_xmit_performance_byte [expr $bac_app_rbytes / $bac_xmit_period]
puts "    "
puts "Conversation application transmission performance by bytes : $con_app_xmit_performance_byte"
puts "Stream application transmission performance by bytes : $str_app_xmit_performance_byte"
puts "Interactive application transmission performance by bytes : $int_app_xmit_performance_byte"
puts "Background application transmission performance by bytes : $bac_app_xmit_performance_byte"
set con_app_xmit_performance_pkt [expr [expr $con_app_rbytes / $udp_con_pktsize] / $con_xmit_period]
set str_app_xmit_performance_pkt [expr [expr $str_app_rbytes / $udp_str_pktsize] / $str_xmit_period]
set int_app_xmit_performance_pkt [expr [expr $int_app_rbytes / $udp_int_pktsize] / $int_xmit_period]
set bac_app_xmit_performance_pkt [expr [expr $bac_app_rbytes / $udp_bac_pktsize] / $bac_xmit_period]
puts "    "
puts "Conversation application transmission performance by packets : $con_app_xmit_performance_pkt"
puts "Stream application transmission performance by packets : $str_app_xmit_performance_pkt"
puts "Interactive application transmission performance by packets : $int_app_xmit_performance_pkt"
puts "Background application transmission performance by packets : $bac_app_xmit_performance_pkt"
}

#Define a 'finish' procedure
proc finish {} {
        global ns tf
        $ns flush-trace
        close $tf
        #Execute trace file process
        set PERL "/usr/bin/perl"
```

```
            set USERHOME [exec env | grep "^HOME" | sed /^HOME=/s/^HOME=//]
            set NSHOME "$USERHOME/ns2/ns-allinone-2.30"
            set XGRAPH "$NSHOME/bin/xgraph"
            set GETSET "$NSHOME/ns-2.30/bin/getset"
            set GETDRT "$NSHOME/ns-2.30/bin/getdrt"
            set EEDELAYS "$NSHOME/ns-2.30/bin/eedelay_s"
            exec $PERL $GETDRT -s 0.0 -d 6.0 -f 1 umts-pdb-out-p.tr > predumtscdrtp.tr
            exec $PERL $GETDRT -s 1.0 -d 7.0 -f 2 umts-pdb-out-p.tr > predumtssdrtp.tr
            exec $PERL $GETDRT -s 2.0 -d 8.0 -f 3 umts-pdb-out-p.tr > predumtsidrtp.tr
            exec $PERL $GETDRT -s 3.0 -d 9.0 -f 4 umts-pdb-out-p.tr > predumtsbdrtp.tr

            exit 0
}

set node_(s1) [$ns node]
set node_(s2) [$ns node]
set node_(s3) [$ns node]
set node_(s4) [$ns node]
set node_(g1) [$ns node]
set node_(g2) [$ns node]
set node_(r1) [$ns node]
set node_(r2) [$ns node]
set node_(r3) [$ns node]
set node_(r4) [$ns node]

$ns duplex-link $node_(s1) $node_(g1) 2.0Mb 1ms DropTail
$ns duplex-link $node_(s2) $node_(g1) 2.0Mb 1ms DropTail
$ns duplex-link $node_(s3) $node_(g1) 2.0Mb 1ms DropTail
$ns duplex-link $node_(s4) $node_(g1) 2.0Mb 1ms DropTail
$ns duplex-link $node_(g1) $node_(g2) 3Mb 100ms UmtsPdbQueue
$ns duplex-link $node_(r1) $node_(g2) 2.0Mb 1ms DropTail
$ns duplex-link $node_(r2) $node_(g2) 2.0Mb 1ms DropTail
$ns duplex-link $node_(r3) $node_(g2) 2.0Mb 1ms DropTail
$ns duplex-link $node_(r4) $node_(g2) 2.0Mb 1ms DropTail
set umtspdbqueue [[$ns link $node_(g1) $node_(g2)] queue]

#Setup UmtsPdbQueue queue parameter
$ns queue-limit $node_(g1) $node_(g2) 40
$umtspdbqueue set c_enq_min 8
$umtspdbqueue set s_enq_min 7
$umtspdbqueue set i_enq_min 6
$umtspdbqueue set b_enq_min 5
$umtspdbqueue set c_enq_max 40
$umtspdbqueue set s_enq_max 38
$umtspdbqueue set i_enq_max 36
$umtspdbqueue set b_enq_max 34
$umtspdbqueue set con_redenque_prob 1.0
$umtspdbqueue set str_redenque_prob 0.9
$umtspdbqueue set int_redenque_prob 0.8
$umtspdbqueue set bac_redenque_prob 0.7
$umtspdbqueue set con_queue_limit 8
$umtspdbqueue set str_queue_limit 7
$umtspdbqueue set int_queue_limit 6
$umtspdbqueue set bac_queue_limit 5

$ns duplex-link-op $node_(g1) $node_(g2) queuePos 0.5

$ns duplex-link-op $node_(s1) $node_(g1) orient up
$ns duplex-link-op $node_(s2) $node_(g1) orient left-up
$ns duplex-link-op $node_(s3) $node_(g1) orient left-down
$ns duplex-link-op $node_(s4) $node_(g1) orient down
$ns duplex-link-op $node_(g1) $node_(g2) orient right
$ns duplex-link-op $node_(g2) $node_(r1) orient up
$ns duplex-link-op $node_(g2) $node_(r2) orient rigth-up
$ns duplex-link-op $node_(g2) $node_(r3) orient rigth-down
```

```
$ns duplex-link-op $node_(g2) $node_(r4) orient down

#Setup a UMTS UDP connection
set udp_s1 [new Agent/UDP/UDPUmtsc]
set udp_r1 [new Agent/UDP/UDPUmtsc]
$ns attach-agent $node_(s1) $udp_s1
$ns attach-agent $node_(r1) $udp_r1
$ns connect $udp_s1 $udp_r1
set udp_con_pktsize 1000
$udp_s1 set packetSize_ $udp_con_pktsize
$udp_r1 set packetSize_ $udp_con_pktsize
$udp_s1 set fid_ 1
$udp_r1 set fid_ 1

set udp_s2 [new Agent/UDP/UDPUmtss]
set udp_r2 [new Agent/UDP/UDPUmtss]
$ns attach-agent $node_(s2) $udp_s2
$ns attach-agent $node_(r2) $udp_r2
$ns connect $udp_s2 $udp_r2
set udp_str_pktsize 1000
$udp_s2 set packetSize_ $udp_str_pktsize
$udp_r2 set packetSize_ $udp_str_pktsize
$udp_s2 set fid_ 2
$udp_r2 set fid_ 2

set udp_s3 [new Agent/UDP/UDPUmtsi]
set udp_r3 [new Agent/UDP/UDPUmtsi]
$ns attach-agent $node_(s3) $udp_s3
$ns attach-agent $node_(r3) $udp_r3
$ns connect $udp_s3 $udp_r3
set udp_int_pktsize 1000
$udp_s3 set packetSize_ $udp_int_pktsize
$udp_r3 set packetSize_ $udp_int_pktsize
$udp_s3 set fid_ 3
$udp_r3 set fid_ 3

set udp_s4 [new Agent/UDP/UDPUmtsb]
set udp_r4 [new Agent/UDP/UDPUmtsb]
$ns attach-agent $node_(s4) $udp_s4
$ns attach-agent $node_(r4) $udp_r4
$ns connect $udp_s4 $udp_r4
set udp_bac_pktsize 1000
$udp_s4 set packetSize_ $udp_bac_pktsize
$udp_r4 set packetSize_ $udp_bac_pktsize
$udp_s4 set fid_ 4
$udp_r4 set fid_ 4

#Setup a UMTS Conversation Application
set umtscapp_s [new Application/UMTSCApp]
set umtscapp_r [new Application/UMTSCApp]
$umtscapp_s attach-agent $udp_s1
$umtscapp_r attach-agent $udp_r1
$umtscapp_s set pktsize_ 1000
$umtscapp_s set random_ false

#Setup a UMTS stream Application
set umtssapp_s [new Application/UMTSSApp]
set umtssapp_r [new Application/UMTSSApp]
$umtssapp_s attach-agent $udp_s2
$umtssapp_r attach-agent $udp_r2
$umtssapp_s set pktsize_ 1000
$umtssapp_s set random_ false

#Setup a UMTS Interactive Application
set umtsiapp_s [new Application/UMTSIApp]
```

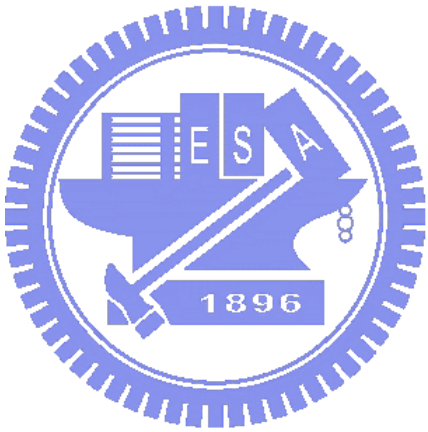


```
set umtsiapp_r [new Application/UMTSIApp]
$umtsiapp_s attach-agent $udp_s3
$umtsiapp_r attach-agent $udp_r3
$umtsiapp_s set pktsize_ 1000
$umtsiapp_s set random_ false

#Setup a UMTS Background Application
set umtsbapp_s [new Application/UMTSBApp]
set umtsbapp_r [new Application/UMTSBApp]
$umtsbapp_s attach-agent $udp_s4
$umtsbapp_r attach-agent $udp_r4
$umtsbapp_s set pktsize_ 1000
$umtsbapp_s set random_ false

#Simulation Scenario
set start_all_time 0.0
set stop_con_time1 7.5
set stop_str_time1 9.5
set stop_int_time1 10.5
set start_con_time1 10.25
set start_str_time1 12.0
set start_int_time1 13.0
set stop_str_time2 14.25
set stop_int_time2 15.75
set stop_con_time2 16.0
set start_int_time2 17.25
set start_con_time2 17.6
set start_str_time2 18.0
set stop_con_time3 20.0
set stop_str_time3 20.6
set stop_int_time3 21.35
set start_str_time3 24.4
set start_int_time3 25.5
set start_con_time3 27.0
set stop_all_time 30.0

set con_xmit_period 0.0
set str_xmit_period 0.0
set int_xmit_period 0.0
set bac_xmit_period 0.0
```




```
  set con_xmit_period [expr [expr $stop_con_time1 - $start_all_time] + [expr $stop_con_time2 -
    $start_con_time1 ] + [expr $stop_con_time3 - $start_con_time2 ] + [expr $stop_all_time -
    $start_con_time3]]
  set str_xmit_period [expr [expr $stop_str_time1 - $start_all_time] + [expr $stop_str_time2 -
    $start_str_time1 ] + [expr $stop_str_time3 - $start_str_time2 ] + [expr $stop_all_time - $start_str_time3]]
  set int_xmit_period [expr [expr $stop_int_time1 - $start_all_time] + [expr $stop_int_time2 -
    $start_int_time1 ] + [expr $stop_int_time3 - $start_int_time2 ] + [expr $stop_all_time - $start_int_time3]]
  set bac_xmit_period $stop_all_time

$ns at $start_all_time "$umtscapp_s start"
$ns at $start_all_time "$umtssapp_s start"
$ns at $start_all_time "$umtsiapp_s start"
$ns at $start_all_time "$umtsbapp_s start"
$ns at $stop_con_time1 "$umtscapp_s stop"
$ns at $stop_str_time1 "$umtssapp_s stop"
$ns at $start_con_time1 "$umtscapp_s start"
$ns at $stop_int_time1 "$umtsiapp_s stop"
$ns at $start_str_time1 "$umtssapp_s start"
$ns at $start_int_time1 "$umtsiapp_s start"
$ns at $stop_str_time2 "$umtssapp_s stop"
$ns at $stop_int_time2 "$umtsiapp_s stop"
$ns at $stop_con_time2 "$umtscapp_s stop"
$ns at $start_int_time2 "$umtsiapp_s start"
$ns at $start_con_time2 "$umtscapp_s start"
```

```
$ns at $start_str_time2 "$umtssapp_s start"
$ns at $stop_con_time3 "$umtscapp_s stop"
$ns at $stop_str_time3 "$umtssapp_s stop"
$ns at $stop_int_time3 "$umtsiapp_s stop"
$ns at $start_str_time3 "$umtssapp_s start"
$ns at $start_int_time3 "$umtsiapp_s start"
$ns at $start_con_time3 "$umtscapp_s start"
$ns at $stop_all_time "$umtscapp_s stop"
$ns at $stop_all_time "$umtssapp_s stop"
$ns at $stop_all_time "$umtsiapp_s stop"
$ns at $stop_all_time "$umtsbapp_s stop"
$ns at 31.0 "$umtspdbqueue printstats"
$ns at 31.0 "output"
$ns at 31.0 "finish"

$ns run
```



Responses to commitee comments

口試委員問題與建議回覆

Prof. Chi-Chun Lo(羅濟群教授)

項次	問題與建議	問題回覆
1.	現行 UMTS 怎麼做？為什麼要用你的方法？你的貢獻在那？如果你已知道應用的 packet type 你就可以選擇最好的方法管理 queuing system？	已請教專家回覆信：All-IP 於 UMTS LTE 才會實現，目前 UMTS 的環境 QoS 有在做 本研究所提出二種佇列空間配置，主要是能夠做到差別性服務的特性，並可實現在不同網路型態上，採用 UMTS 是因為 UMTS 有明確的定義差異性服務等級，可以作為差別性服務的代表 我的貢獻在於只需要小成本的方式在 Router 上採用這個機制，所得的結果可與 DiffServ Domain 近似甚至有些服務類型優於 DiffServ，節省許多建置上的成本 真實世界的網路環境，並無法事先預知何種 Packet Type 會送達，所以無法事先選擇最好的 Queuing System 管理，因此選擇最好的管理方式常是兩難的情況

Prof. Taoi Hsu(徐道義教授)

項次	問題與建議	問題回覆
1.	應參考 Benchmark 的演算機制進行效能的比較優劣？	模擬後比較結果，請參考表 1, 2 和 3.

Prof. Chyan Ynag(楊 千教授)

項次	問題與建議	問題回覆
1.	程式碼也需附在附錄中	論文初稿中已修正

Prof. Duen-Ren Liu(劉敦仁教授)

項次	問題與建議	問題回覆
1.	與現有的 DiffServ 架構之效能比較	如上，同徐教授問題

Prof. Tsung-Li Wu(吳宗禮教授)

項次	問題與建議	問題回覆
1.	DQB 與 OQB 配置在 Edge Router 或 Core Router 上？	在 Internet 環境中，Core Router 在其他視界中就成為 Edge Router，所以，並不特定是配置在那種 Router 上，但以邏輯上而言是 Core Router，Edge Router 有另外負責的工作，如分類、標記…
2.	提出新的機制？改善 Disorder 及 Starvation 的問題，放棄 DQB 和 OQB？	目前已在進行中，修改程式調整參數，以求解決方法： Disorder 問題只會發生在 OQB 中，每次保證空間內的封包要送出佇列時，先檢查 Overflow buffer 的封包是否相同？！接著再比對封包的序號作排序 Starvation 的問題發生在 DQB 中：後續要用調整參數值的方法以求飢餓的狀況不會產生
3.	為什麼選擇 DiffServ？	IntServ 和 DiffServ 是目前 IP QoS 兩大主流，而 IntServ 有建置上的困難與成本的負荷，且需要所有路徑上的 Router 支援 RSVP 協定；因 DiffServ 不需要維護每個 flow 也不需要通知路由器為其預留資源，簡化了控制信號工作，並可對 IP 表頭內 TOS 段來區別不同的的優先等級以提供不同的服務。 3GPP 已定義優先順序類別映射至 DiffServ 中的 PHB 各類設定，所以本研究選擇 DiffServ 模式，如：表 4 所示.
4.	Mapping 至 Fig.4 UMTS QoS Architecture 研究上的 QoS 是那一段？	提供 CN Bearer Service/Backbone Bearer Service 之間的 QoS，因本研究主要是在核心網路中運作
5.	研究上提出的 Queuing Scheme 和 DiffServ Router 是否不同？	本研究的 Queuing Scheme 是在 Router 上運作的機制，研究模擬所得的結果與 DiffServ 可達近似的效果 DiffServ 是一個架構由許多 Router(Edge Router, Egress/Ingress Router, Core Router)組成，可能是一個 ISP 或一個企業網路，建置成本相對會比較高
6.	AQM/RED 能發揮作用是因為搭配 TCP 壅塞避免的機制，才會有效果？對於 UDP 的封包是否就無效？對於	AQM/RED 由於是運作在網路層，故不論 TCP 或 UDP 的封包都可以利用 RED 機制發揮壅塞避免的功能，RED 的壅塞避免控制是以佇列空間大小為基準，決定是否丟棄封

<table>
<tr><td></td><td>你提出的研model中是否也有相同效果？</td><td>包，並非特定封包(TCP)才能發揮效能

在 DiffServ 架構中，壅塞避免也是利用 AQM 中的 WRED 機制所實作的</td></tr>
<tr><td>7.</td><td>封包 size 為什麼要定義 100、500、1000？</td><td>本研究利用三種封包的 SIZE 是大、中、小封包的一種代表性，因為目前網路上的封包有大有小，沒有一定的 size，而我們也無法預測進來的封包有多大？是何種類型封包？

目前多媒體類型的封包會因採用不同的壓縮標準而有不同的封包大小，以 VoIP 而言，壓縮的標準有 G.723、G.729 與 G.711 等等。不一樣的語音壓縮標準代表著不同的語音封包大小形式，同時也代表著不同的語音壓縮比率，佔用不一樣的頻寬，會有不一樣的延遲時間，再封裝下列 header 後，於網路中傳送。

N. V. Lopes, M. J. Nicolau and A. S., “Efficiency of PRI and WRR DiffServ Scheduling Mechanisms for Real-Time Services on UMTS Environment”, 2nd IFIP International Conference on New Technologies, Mobility and Security (NTMS 2008), Tangier, Morocco, Nov 5-7, 2008, ISBN: 978-2-9532443-0-4, pp 173-178. This study shows traffic characteristics the following table 5.

Table 5. Traffic characteristics

<table>
<tr><th></th><th>VoIP</th><th>Video Streaming</th><th>Web Browsering (HTTP)</th><th>FTP</th></tr>
<tr><td>Application Protocol</td><td>Exponential</td><td>CBR</td><td>HTTP</td><td>FTP</td></tr>
<tr><td>Transport Protocol</td><td>UDP</td><td>UDP</td><td>TCP</td><td>TCP</td></tr>
<tr><td>Packet Size (bytes)</td><td>372</td><td>540</td><td>500</td><td>500</td></tr>
<tr><td>Rate (bps)</td><td>12.2(x7 source)=85</td><td>47.7(x7 source)=333</td><td>160</td><td>484</td></tr>
<tr><td>Rate (%)</td><td>8</td><td>31</td><td>15</td><td>46</td></tr>
<tr><td>Interleaving(ms)</td><td>20</td><td>2</td><td>7</td><td>20</td></tr>
</table>

Voice: 372bytes: 10 frames per packets: 20(IPv4)+8(UDP)+12(RTP)+332(max RTP payload for 10 AMR frames). It can be modelized by the two Markov process as suggested by ITU-T.

Video: 540bytes: 20(IPv4) + 8(UDP) + 12(RTP) + 500(max AMR RTP payload). The most of commercial videos are CBR.

Web Browser: 500bytes: Based on large packet sizes used in Internet.

FTP: FTP sessions behave similar to HTTP requests but without the page abstraction level.</td></tr>
</table>

8.	不同資料型態的各種特性於封包中是否也能突顯？	由於本研究是模擬而非仿真，所以要真正切合實際環境中的封包特性的確有其困難度 各種 Traffic Pattern，NS2 有提供並可於模擬時寫在 TCL 腳本中
9.	DQB 與 OQB 應用在 UMTS 之 uplink/downlink Channel 中是否會有不同效果？	TE(終端設備端)才有 Downlink 與 Uplink 的問題，核心網路中沒有 Downlink 與 Uplink 的問題
10.	TCP on UMTS 之一般效能如何？如果不佳時(很多 retransmission 時)怎麼辦？	TCP 的效率問題不管在何種網路架構中，都有可能會發生，不會因為是 UMTS 就不會有 Retransmission 問題，本研究的重點不在於 TCP 傳輸層，而是在網路層運作
11.	Table 4 中，有過其他安排？為什麼要這麼設定？	由多組想定實驗所得並配合各應用服務的優先等級順序條件下，所得 Table 4 設定是最佳參數值組

Table 1. A Summary of simulation parameters

Parameter dimensions	***Setting values***				
Packet size	500 bytes				
Two-phase queueing buffer allocation scheme settings	**UMTS Application**	**GBS**	**MIN**	**MAX**	**PEP**
	Conversational	8	8	40	1
	Streaming	7	7	38	0.95
	Interactive	5	5	35	0.85
	Background	2	2	31	0.7
UMTS traffic bandwidth requirement	1.0MB	***Backbone bandwidth in the UMTS core network***			2.0MB
Queueing buffer allocation	**DQB, OQB, DiffServ**				
Traffic transmission pattern	Continuous		***Simulation time***		60 Sec

Legends: **GBS**: Guaranteed buffer size, **PEP**: packet enqueueing probability,
MIN: minimum limit, **MAX**: maximum limit

Table 2. Average packet jitter/delay statistic between DQB, OQB and DiffServ

UMTS Traffic	*Conversational*		*Streaming*		*Interactive*		*Background*	
Allocations	**jitter**	**delay**	**jitter**	**delay**	**jitter**	**delay**	**jitter**	**delay**
DQB	0.000016	0.22751	0.002112	0.158768	0.000164	0.153704	0.020889	0.201037
OQB	0.00162	0.164011	0.002518	0.164316	0.002047	0.186736	0.002037	0.171697
DiffServ	0.000021	0.311937	0.000739	0.267067	0.001391	0.267066	0.0125	0.265889

Legends:
Unit : ms, **PS :** packet size, **B :** bytes

Table 3. Packet enqueuing/dequeuing statistic

Packet Size: 500 bytes	60 seconds	*Conversational*	*Streaming*	*Interactive*	*Background*
Arrival packets	DQB	15001	15001	15001	15001
	OQB	15001	15001	15001	15001
	DiffServ	14999	14999	14999	14999
Enqueued packets	DQB	15001	9007	6004	28
	OQB	15001	9012	4016	2009
	DiffServ	14999	11093	3880	45
Dequeued packets	DQB	15001	9007	6004	28
	OQB	15001	9012	4016	2009
	DiffServ	14999	11093	3880	45
Dropped packets	DQB	0	5994	8997	14973
	OQB	0	5989	10985	12992
	DiffServ	0	3906	11119	14954
Packet dequeued ratio (%) (Throughput)	DQB	100	60.04266382	40.0239984	0.186654223
	OQB	100	60.07599493	26.77154856	13.3924405
	DiffServ	100	73.95826388	25.86839123	0.300020001
Packet dropped ratio (%)	DQB	0	39.95733618	59.9760016	99.81334578
	OQB	0	39.92400507	73.22845144	86.6075595
	DiffServ	0	26.04173612	74.13160877	99.69998

Table 4. QoS mapping between DiffServ PHBs and UMTS Service class

Application Type	*UMTS Service Class*	*DiffServ PHB*
VoIP	Conversational	EF (Expedited Forwarding)
Video Streaming	Streaming	AF1X (Assured Forwarding)
Web, Telnet	Interactive	AF2X
FTP, email	Background	BE (Best Effort)