ADAPTIVE BANDWIDTH MANAGEMENT FOR UMTS NETWORKS

A DISSERTATION

Submitted in partial fulfilment of the requirements for the award of the degree

of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

DEVENDRA SINGH

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE—247 667 (INDIA)

FEBRUARY 2003

CANDIDATE'S DECLARATION

I hereby declare that the work which is being presented in the dissertation entitled " **Adaptive Bandwidth Management For UMTS Networks"** in partial fulfilment of the requirement of the award of degree of **Master of Technology,** in Computer Science and Engineering, submitted in the department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee, is an authentic record of my own work carried out from Aug 2002 to Feb 2003, under the guidance of **Dr. Mohan Lal**, *Asst. Professor*, New Computational Facility, and **Dr. Manoj Misra,** *Associate Professor*, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee.

The matter embodied here, has not been submitted by me for the award of any other degree or diploma.

Place: Roorkee

Date: 27|02\03.

[illegible]

Devendra Singh

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

[illegible]

(Dr. Mohan Lal)
Asst. Professor,
New Computational
Facility,
IIT Roorkee,
Roorkee-247667.
Date :
Place : Roorkee

[illegible]

(Dr. Manoj Misra)
Associate professor,
Dept. of Electronics &
Computer Engineering
IIT Roorkee,
Roorkee-247667.
Date :
Place : Roorkee

ACKNOWLEDGEMENT

I take this opportunity to thank all the magnanimous persons who rendered their full support to my work directly or indirectly. I would like to express my deep sense of gratitude to my guide **Dr. Mohan Lal,** *Asst. Professor*, New Computational Facility, and **Dr. Manoj Misra,** *Associate Professor,* (Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee.) for their valuable suggestions and cooperation throughout this dissertation apart from providing useful material.

My special thanks to **Dr. A. K. Sarje,** Head of Department, Department of Electronics and Computer Engineering, and **Dr. R. P. Agarwal** (Professor and former H.O.D. E & CE Department.) for their constant support and advice, which has helped me in completion of the dissertation.

I am thankful to my friends for their timely suggestions, cooperation and help. My deep appreciation goes to my parents for the encouragement and support; at last I gratefully acknowledge my deep indebtedness to all other persons who helped me during the whole period of the work.

Devendra Singh

DEVENDRA SINGH

ABSTRACT

Universal Mobile Telecommunications system (UMTS) provides the technical foundation to integrate all kind of services like speech, data, audio, video, e-mail, web browsing etc. through the mobile hand set. The objective here is to improve bandwidth utilization and quality of service of Universal Mobile Telecommunications System.

To improve bandwidth utilization of Universal Mobile Telecommunications System performance management information base (P-MIB) has been implemented which dynamically adjusting the packet scheduler and admission controller. The P-MIB also controls the system parameters to improve overall quality of service. The key contribution of this dissertation is to compare the performance of two packet scheduler i.e., weighted round robin scheduler and weighted fair queuing schedulers. The two packet schedulers are implemented so that maximum number of packets are transmitted in less time unit, which in turn maximizes the bandwidth utilization.

The implementation is done in c++ language on Pentium IV system under windows-98 environment.

CONTENTS

INTRODUCTION

1.1 UMTS Networks

Universal mobile telecommunications system (UMTS) provides the technical foundation for integrating the currently separate worlds of mobile and fixed telecommunications services into a unified digital data environment. Universal mobile telecommunications system (UMTS) is characterized by a migration from voice-only to integrated services networks. Thus, applications such as e-mail, web browsing, and corporate local network access, as well as video conferencing, e-commerce, and multimedia can be supported over wireless data channels.

1.2 UMTS network Architecture

The UMTS network architecture is divide into following parts

- UMTS core network (CN)
- UMTS access network (UTRAN)
- Node B.
- User equipment.

The UMTS core network is responsible for handling circuit switched connections and tunneling packet switched data to public networks (i.e., internet)
The UMTS access network i.e., the UMTS terrestrial radio access network consists of a set of Radio Network Controller (RNC), Radio Base Station (RBS) [1][2][14].

The main task of the RNC is to manage Radio Access Bearers (RABs) for user data transport, manage and optimize the radio network resources and control mobility, while the RBS provides the actual radio resources and maintains the radio links.

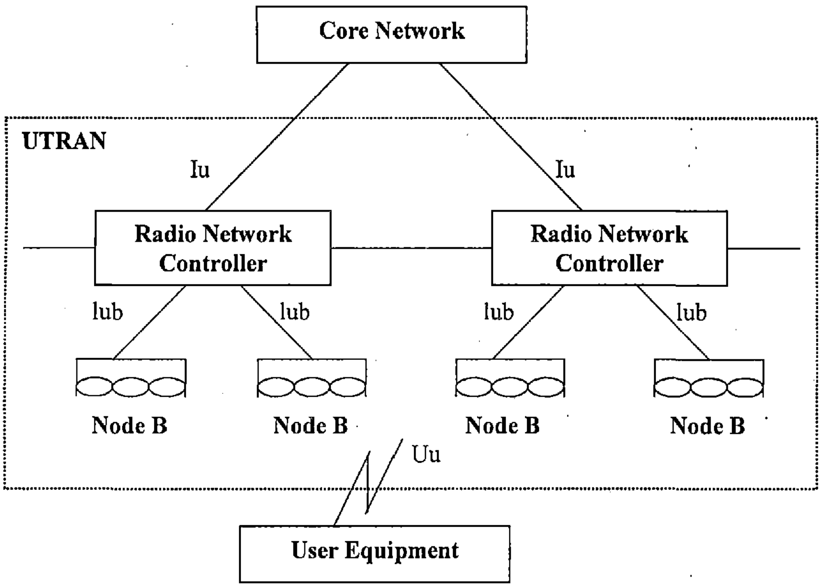


Fig 1 Architecture of UMTS.

The RNC is connected to the CN via the Iu interface and the User Equipment (*UE*) is connected to the RBS via the Uu (radio) interface.
Internally within RAN, the RNCs are interconnected via the Iur interface to support smooth handover for MS leaving the area covered by the serving RNC and entering the area of a drifted RNC, as shown in figure 1[1].

The Node-B is physical unit of radio transmission/reception with cells. The Node-B is the equivalent of BTS in GSM and typically serves a cell site. Several Node-Bs are controlled by single RNCs over the Iub interface. The main task of Node-B is to measure quality and strength of the connection and determines the error rate [1][2][12][14].

1.3 Quality Of Service Classes

Universal Mobile Telecommunications system provides four different classes of QoS[1][7].

1. Conversational class.
2. Streaming class.
3. Interactive class.
4. Background class.

The main distinguishing factor between these QoS classes lies in the delay sensitivity of the traffic. The conversational class is meant for traffic, which is very delay sensitivity while the background class is the most delay insensitive traffic class. Conversational and streaming classes are mainly intended to be used to carry RT traffic. A conversational RT traffic stream is characterized by requiring low transfer delay and small delay jitter because of conversational nature of the stream. The streaming traffic class consists of one-way RT traffic streams, e.g., viewing video clips or audio clips. Interactive class and background class are mainly meant to be used by traditional Internet applications like WWW, e-mail and FTP. The main difference between the interactive class and the background class is that interactive class is mainly used for applications such as, interactive Web browsing, while background class is meant for background download of e-mail or background file downloading. Traffic in the interactive class has higher priority than background traffic. Thus background application use transmission resources only when interactive application do not need them [6][16].

1.4 Statement Of Problem

In this report the problem of improving bandwidth utilization of Universal Mobile Telecommunications system is addressed and performance management information base (P-MIB) has been designed which improves overall QoS of

UMTS by controlling system parameters i.e., threshold, bandwidth portions and queuing weights.

1.5 Organization of the Dissertation

The dissertation is organized as follow.

Chapter 2 describes the framework for the adaptive performance management comprising of an admission controller and packet scheduler. The two type of packet scheduler i.e., weighted round robin and weighted faire queuing are being described under packet scheduler.

Chapter 3 provides the framework for online performance monitoring and adaptively adjusting system parameters.

Chapter 4 describes the simulation strategy.

Chapter 5 presents the simulation results to show the benefit of employing the framework for adaptive performance management and packet scheduling strategy's.

Chapter 6 Finally concluding remarks and scope for future work is given.

APPENDIX A: flow charts.

APPENDIX B: source code.

FRAMEWORK FOR ADAPTIVE PERFORMANCE MANAGEMENT

2.1 Admission control

The proposed framework distinguishes three different types of services

- Circuit-switched services.
- Packet switched real time services.
- Non- real time services.

In general circuit switched services are voice calls from a mobile station. Real time services correspond to UMTS conversational and streaming class and non-real time (NRT) services correspond to the UMTS interactive and background class. The bandwidth available in a cell must be shared by calls of different service classes and the different service requirements have to be met.

So whenever a mobile session starts, user has to specify its traffic characteristics and desired performance requirements called as QoS profile. Then the admission controller decides to accept or reject the users request based on QoS profile and the current network state e.g., given by queue length. The purpose of admission controller is to guarantee the QoS requirements of the user who request admission while not violating QoS profile of already admitted users. The call admission criteria will be different for each service class. Admission control of RT session is based on a QoS profile that specifies a guaranteed bit rate that should be provided to the application to work proper. If the network cannot satisfy the desired bandwidth requirements the corresponding admission request is rejected [1][8][13][14].

2.2 Adaptive Bandwidth Partitioning

The partitioning of the available bandwidth is performed to meet the QoS requirements of the three service classes:

- Voice calls.
- RT session.
- NRT sessions.

Let B be the overall bandwidth available in one cell. A portion b_h of the bandwidth B is reserved for handover calls from neighboring cells in order to reduce handover failures. A portion b_p is reserved for RT and NRT data packets, i.e., packets only.

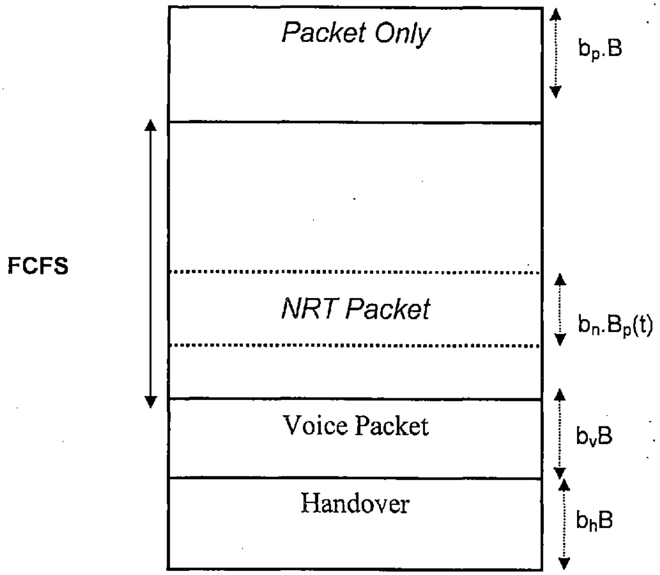


Fig. 2.1. Adaptive bandwidth partitioning of the available bandwidth.

The remaining bandwidth $(1-b_h-b_p)B$ can be allocated "on demand" by voice calls and data sessions respectively. Because in future UMTS networks voice calls will still play a major roll in bandwidth requirements. So a portion b_v of overall bandwidth is allocated for voice calls. The remaining bandwidth is allocated on a first-comes first-served (FCFS) basis to voice calls or RT sessions. In order to give NRT traffic a certain amount of bandwidth, a portion b_n of the bandwidth actually available for packet data is reserved for non-real-time packets. Let $B_v(t)$ be the bandwidth reserved for all voice calls at a certain time t, then for packet data bandwidth of size $B_p(t)=B - B(t)$ is available. Different call type arriving in the cell and the corresponding condition under which these call are admitted are as follows:

1. For RT users, admission is based on the availability of the guaranteed bandwidth specified in the QoS profile. Let $B_r(t)$ be the bandwidth already allocated for RT traffic at time t and let B_r be the bandwidth required by the user who requested admission. The user will be admitted according to the bandwidth partitioning [1].

   For Real Time Call

   - $B_r + B_r(t) \leq (1-b_h)B - B_v(t)$.
   - $B_r + B_r(t) \leq (1-b_n)(B - B_v(t))$.

   That is after call admission the handover bandwidth is still available and a portion b_n of the overall packet bandwidth $B_p(t)$ is also still available for NRT sessions

2. For Voice Call
   - New Voice Calls With Bandwidth requirement B_v will be admitted if there is enough bandwidth in voice packet region (i.e. $b_v.B$).
   - If voice call can be accommodated in the first come first serve area without violating bandwidth requirement of ongoing calls.
3. For NRT Call
   - For NRT session, the admission is based on availability of buffer space in NRT Queue.

2.3 Packet Scheduler

Data packets from various connections are queued until bandwidth is available for transmission. Packet scheduler controls the order in which packets are served and how packets in transfer share the available bandwidth. Data packet that arrive at the radio network controller are organized in two distinct queues i.e., real-time queue and non-real-time queue. A transfer queue is also implemented that contains the packets actually in transfer. The available bandwidth capacity is shared over the packets in the queue according to their QoS requirements and bandwidth partitioning. Each time one of the following events occurs the available bandwidth is newly assigned to packets in transfer by a packet scheduler:

Events

- Admission of new voice call or handover.
- Termination of a voice call due to call termination or handover.
- Transfer of a RT or NRT data packet is finished.
- Arrival of a RT packet of user I at the radio network controller with no RT packet of user I waiting in the RTQ or been in transfer,
- Arrival of a NRT packet at a radio network controller with no NRT packet waiting in the NRTQ and still bandwidth capacity available for NRT packets.

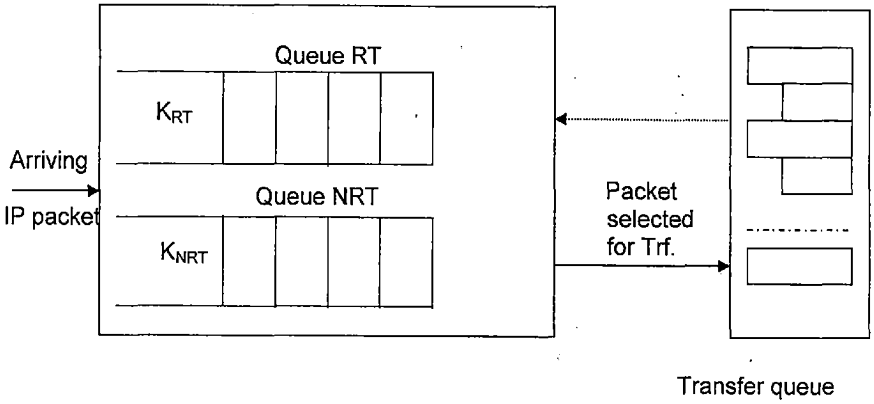


Fig. 2.2 RTQ, NRTQ and transfer queue.

In order to distinguish different priorities for NRT traffic, a weighted Round Robin scheduler or more complex scheduling strategies like weighted fair queuing (WFQ) has been implemented. Whenever the packet scheduling is initiated due to the occurrence of one of the events stated above, the available bandwidth is newly assigned to packets in transfer [1][5].

Let t be the point in time when the packet scheduling is initiated. Voice calls are assumed to be circuit switched. Therefore, each voice call allocates a fixed amount of bandwidth during its lifetime. The bandwidth requirements $B_r(t)$ needed to satisfy the guaranteed bit rate for RT users is computed. If a portion b_n of the remaining packet bandwidth is not anymore available for NRT packets then RT sessions have to be degraded. This can only happen if bandwidth requirements for RT packets are so exhaustive that they occupy the voice packets areas and an additional voice call is admitted in the cell. The degradation of RT sessions if performed stepwise i.e., in each degradation step all RT sessions are degraded to a specified level before starting the next degradation step if necessary. After assigning bandwidth to RT sessions the remaining bandwidth is allocated to NRT traffic. If still bandwidth available then degraded RT sessions can be increased to their guaranteed bandwidth again.

2.4 Weighted Round Robin Scheduling

In this type of scheduling strategy packets from different connection arrives and are queued until bandwidth is available for transmission. The queues are treated as a circular queue. A small unit of time, called a time quantum or time slice, is defined. The scheduler goes around the queue, allocate the bandwidth to each connection for a time interval up to a quantum in length [13][15].

Implementation

- **On packet arrival**
  - Set the time quantum.

- Pick the first packet from the queue, allocate the bandwidth till the time quantum is completed.
- If the packet size is larger than the time quantum, store the source and destination address of the packet and put the packet at the tail of queue.
- If the packet size is smaller than the time quantum, then select the next packet of transmission.

Disadvantage

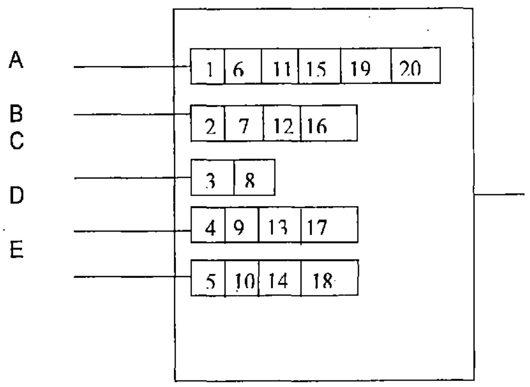
- It is better if packets are of same length.
- It gives more bandwidth to calls that uses large packets than to calls that uses small packet.

2.5 Weighted Fair Queuing

In weighted fair queuing instead of a packet-by-packet round robin, it scans the queue repeatedly, byte-for-byte, until it finds the tick on which each packet will finished. The packet are then sorted in order of their finishing times and sent in that order [13][15].

Implementation

- **On packet arrival**
  - Use source + destination address to classify it and look up finish number of last packet served (or waiting to be served).
  - Compute finish time.
  - Insert in priority queue sorted by finish times.
  - If no space, drop the packet with largest finish time.
- **On service completion**
  - Select the packet with the lowest finish time.



Packet	Fin. Time
C	8
B	16
D	17
E	18
A	20

Fig. 2.3. Finishing time for five packets.

In fig 2.3 Packet of length 2 to 6 bytes are specified. At clock tick 1, the first byte of the packet on line A is sent. Then goes the first byte of the packet on line B, and so on. The first packet to finish is C, after 8 ticks. The sorted order is given in fig(2.3). The packets will be sent in the order listed, from C to A [13]. The main objective of including scheduler is that maximum number of packets scheduled in minimum amount of time.

For both type of scheduling scheme average turn around time is calculated. Average Turn around time is the time taken to complete the transmission of packet.

$$A_{rg} = \frac{S_m}{N}$$

Where

A_{rg} = Average turn around time

S_m = sum of time taken by each packet to transmit.

N =Number of packets

A comparison is done for both the scheduling schemes, to identify which scheduling strategy is better so that bandwidth utilization is maximized.

PERFORMANCE MANAGEMENT INFORMATION BASE

3.1 System Architecture For Adaptive Performance Management

The system architecture for adaptive performance management is being divided into following parts

- Online performance monitoring
- System parameters
- Performance management

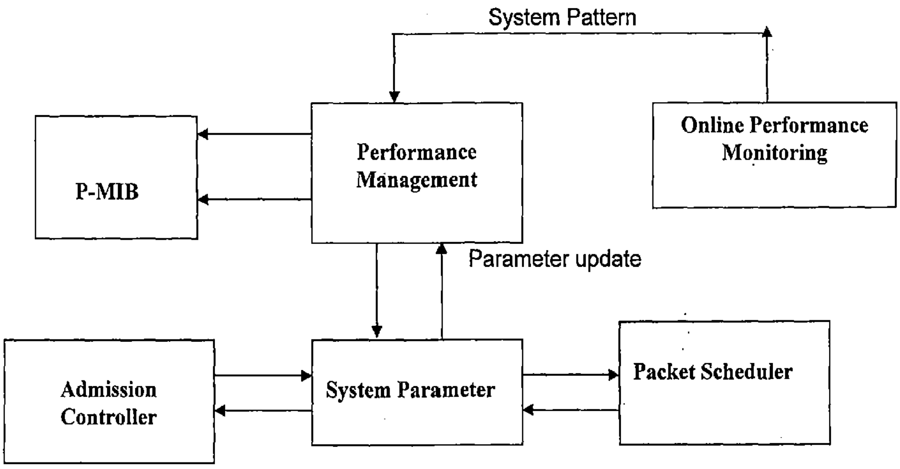


Fig. 3.1. System architecture for adaptive performance management.

To maximize QoS for the mobile users, a performance management entity has to be introduced in a radio network controller that is responsible for corresponding transceiver stations (i.e., Node B elements). Furthermore, a radio network controller has to be extended by an online performance measurement component that derives performance measures in a certain time window (e.g., handover failure probabilities of mobile users). These performance measures form a system pattern. The system pattern is submitted in fixed time intervals to the performance management entity, which subsequently updates the system parameters (i.e., parameters of traffic controlling components like the admission controller and packet scheduler). The update of system parameters is made as specified in a P-MIB [1].

System parameters, which are adjusted by the performance management entity, comprise of

1. Bandwidth portions i.e., handover bandwidth, real-time bandwidth non-real-time and voice bandwidth.
2. NRTQ threshold (portion η of buffer size),
3. Queuing weights wi for NRT packets with priority i.

3.2 Online Performance Monitoring

The online monitoring of QoS measures is performed by a sliding window technique as depicted in Fig. 3.2. The width of the sliding window depends on the number of relevant events that occur according to a performance value (e.g., NRT packet arrivals are relevant events for computing PLP). The upper part of Fig. 3.2 shows the sliding window at a certain time point t_0. Assuming that at time t_1 the next relevant event occurs the sliding window moves in time as shown in the lower part of Fig. 3.2. After a certain number of relevant events are occurred a system parameter update is performed based on the performance measure derived from the sliding window (e.g., update of g according to PLP derived from sliding window). To get expressive performance measures the sliding window should not be too small. As a certain number of events representing the history of the performance value have to be considered [1][8][13].

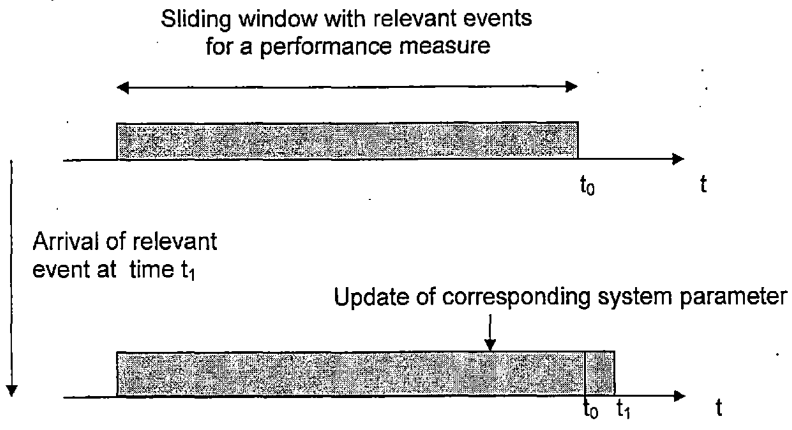

Sliding window with relevant events for a performance measure
t_0
t
Arrival of relevant event at time t_1
Update of corresponding system parameter
t_0 t_1
t

Fig. 3.2. Online performance monitoring

3.3 Updating Of System Parameters

The UMTS system parameters can be updated by monitoring QoS measures, which immediately affect these parameters. The QoS measures are handover failure probability (HFP) and call/session blocking probability (CBP) to voice calls and RT sessions respectively. The packet loss probability of the NRTQ is abbreviated by PLP. The average number of active NRT sessions with priority 1, 2, and 3 is denoted by NRT1, NRT2, and NRT3. Determining the updates for the system parameters, i.e., determining $b_h^{(new)}$ and $\eta^{(new)}$, and the updated queuing weights $w_1^{(new)}$, $w_2^{(new)}$, and $w_3^{(new)}$can be performed based on the dependencies (1)–(3) to the corresponding old values and the actually observed QoS measures HFP, CBP, PLP, NRT1, NRT2, NRT3. That is:

1. $b_h^{(old)}$, HFP, CBP $\rightarrow b_h^{(new)}$
2. $\eta^{(old)}$, PLP$\rightarrow \eta^{(new)}$

3. NRT1, NRT2, NRT3→ $w_1^{(new)}$, $w_2^{(new)}$, $w_3^{(new)}$

The update of system parameters is performed as follow [1]

After deriving the factors K_{HFP}, K_{CBP}, and K_{PLP} (which represents handoff failure probability, call blocking probability and packet loss probability), update the bandwidth partition and threshold value according to dependencies (1) and (2)

- $b_h^{(new)}=k_{HFP}.k_{CBP}.b_h^{(old)}$
- $\text{Threshold}^{(new)} = k_{PLP}.\text{Threshold}^{(old)}$

The update of the queuing weights i.e., determining $w_1^{(new)}$, $w_2^{(new)}$, and $w_3^{(new)}$ is made according to the measured average number of NRT sessions belonging to priorities 1, 2, and 3 in the cell.

The queuing weights are updated according to (3) dependency.

- $w_1^{(new)} = \frac{4.\sqrt{NRT_1}}{W}$. $w_2^{(new)} = \frac{2.\sqrt{NRT_2}}{W}$. $w_3^{(new)} = \frac{1.\sqrt{NRT_1}}{W}$

  $W = 4.\sqrt{NRT_1} + 2.\sqrt{NRT_2} + 1.\sqrt{NRT_3}$

SIMULATION STRATEGY

4.1 System Model

The simulation model consists of a cell cluster comprising of seven hexagonal cells. When a mobile user starts a new session, the session is classified as voice-, RT, or NRT session, i.e., with the session the user utilizes voice-, RT, or NRT services. The voice calls are assumed to be circuit-switched connections that require a constant amount of bandwidth. Users have to specify the QoS profile for RT and NRT sessions. For RT sessions the two QoS profile defines, i.e., a low bandwidth profile comprising of a guaranteed bit rate corresponding to streaming audio and a high bandwidth profile comprising of a guaranteed bit rate corresponding to streaming video. Before a mobile user can start a new session, user has to pass the admission controller. The amount of time that a mobile user with an ongoing session remains within the cell is called dwell time. If the session is still active after the dwell time, a handover toward an adjacent cell takes place [3][4][5]. Thus, in the simulation environment, a session of a mobile user is completely specified by the following parameters: service class packet arrival process, dwell time and QoS profile.

Simulation Model For Weighted Round Robin Scheduler

For modeling weighted round robin scheduler two queues are considered one for real-time packets and another for non-real-time packets. Packets from different connection arrive at radio network controller, which are queued until bandwidth is available. For performing weighted round robin scheduling a time quantum is specified and weighted round robin scheduling is initiated whenever following type of event occurs i.e., new call arrival, handoff, completion of NRT and RT services. The scheduler picks the packet from the queue and checks whether there is enough bandwidth or not. If bandwidth is available packets are given bandwidth for transmission. Every packet is transmitted according to the time quantum, if the time quantum expires before packet transmission that packet is placed at the end of queue otherwise new packet is selected from the queue. Priorities are assigned according to weights i.e., any packet having higher weight have higher priority. Packet with higher priority are served first.

Simulation Model For Weighted Fair Queuing

In weighted fair queuing simulation packets are scanned byte-by-byte and their finish time is calculated. Finish time is the time taken by packet to finish transmission. A queue is maintained which store the packets according to their finish time. The packet at the head of queue will have minimum finish time while the packet at the end will have maximum finish time. The packets are transmitted from the queue i.e., the first packet to be transmitted will have minimum finish time. The weights are assigned for priorities. If packet have higher priority, than at one clock tick more than one byte of that packet is transmitted.

4.2 Simulation Parameters And Assumptions

Simulation parameters are of fixed and adaptive type. Fixed parameters are those parameter that remain static through out the simulation.

Fixed parameters are as follows:

Available bandwidth in one cell, B	7680 kbps
RTQ buffer size, K_{RT}	1000 packets
NRTQ buffer size, K_{NRTQ}	1000 packets

Adaptive parameters are those parameters that can be adjusted by performance management information base.

Adaptive parameters are as follows:

Handover bandwidth	5%
Voice call bandwidth	10%
Data packet bandwidth	20%
FCFS	65%
Bandwidth for NRT packets	10%
NRTQ threshold	90%

- The amount of time a mobile user with ongoing sessions remains within the cell is modelled by lognormal distribution.
- The duration for voice calls and RT sessions is assumed to be exponentially distributed.

4.3 User Inputs

The software offers a certain degree of flexibility by allowing user entry of the following simulation parameters

1. Service Class i.e., whether user wants to use voice, real-time or non-real-time services.
2. Bandwidth required by each type of service.
3. Quality of service profile for each type of service.
4. Time quantum for weighted round robin scheduling.

4.4 Program Structures And Classes:

Structures:

List :This structure is used to store packets in a first in first out manner.

Attributes:

Next : next part is a pointer to next node.

Calls :Before a mobile session starts user has to specify the call type i.e., whether voice call, RT session or NRT session.

Attributes :

Type : which specify the type of call i.e., whether voice call, RT session or NRT session.

atime : it specify the duration of the on going session.

Handover :When user moves from one cell to another, users information has to be passed on.

Attributes :

Type	: This specify the type of handover i.e, whether it voice handover or real time handover.
Priority	: This is being used to assign priority real time handover over voice handover.

Classes :

Queue	:This class is being used to store real-time and non real-time packets.
Attributes :	
Front	: The first item that will be removed from the queue.
Rear	: The last item in the queue, that is, the one most recently added.
qptr	: qptr is a pointer to the queue that tracks the front and rare position of the queue.
Operations:	
Empty	: Checks whether queue is empty or not.
Add	: This function is used to added item to the queue.
Remove	: Removes the item from the queue.

Events	: This class is used to create object of events.
Attributes :	
total_calls	: Total number of calls.
blocked	: Total number of blocked calls.
ho_success	: Number of handover success.
ho_fail	: Number of handover failure.
next_call	: Identify the next call
next_handover	: This attribute is used for next handover type.
ho_delay	: Handover delay.
busy_channels	: Identify whether the channel is busy or not.
next_event_type	: Identify the event type.

max_q_len	: Identify the maximum queue length.
Operations:	
Events	: Initialise all the parameters.
new_call	: This function is used when ever new call arrives and identify the traffic characteristic.
new_handover	: This function checks for new handover call.
release_channel	: This function release the bandwidth allocated after the completion of service.
q_scan	: This function scans the queue for packets.

Simulation	: This class is used to perform the simulation.
Operations:	
Simulation	: This function performs the simulation according to scheme type i.e., weighted round robin or weighted fair queuing.
Traffic	: This function generate the traffic.
Save	: This function stores the result in a file.
Load	: This function calculate overall load in the system.
Start	: This function identify the next event type i.e., new call, handover or channel release.

Functions:

Mobile_subchoice	: when ever a mobile session starts it has to specify its traffic characteristic i.e, voice call, real-time session or non-real-time session.
Attributes :	
Vbwidth	: Bandwidth for voice calls.
Rtbwidth	: Bandwidth for real-time sessions.
Ntbwidth	: Bandwidth for non-real-time sessions.

Vadmission_check : Checks the required voice bandwidth against the available bandwidth. If bandwidth is available the request is accepted otherwise rejected.

Attributes :

Vbwidth	: Bandwidth for voice calls.

rtadmission_check: Checks the required real-time bandwidth against the available bandwidth. If bandwidth is available the request is accepted otherwise rejected.

Attributes :

Rtbwidth	: Bandwidth for real-time sessions.

ntadmission_check : Checks the required non-real-time bandwidth against the available bandwidth. If bandwidth is available the request is accepted otherwise rejected.

Attributes :

Ntbwidth	: Bandwidth for non-real-time sessions.

byte_scan : It scans packet byte_by_byte, calculate their finish time and sort them in ascending order according to their finish time.

Attributes :

Pkt_size	: Size of packet.

P_mib : It is a performance management information base which regularly checks packet loss probability, call blocking probability and handoff failure probability and according to these probabilities it generates new performance controlling parameters i.e, new handover bandwidth, new threshold, new weights.

Attributes :

HANDOFF_FAIL_PROB	: Handoff failure probability.
PKT_LOSS_PROB	: Packet loss probability.
CALL_BLOCK_PROB	: Call blocking probability.

NEW_THRESHOLD	: New threshold value.
w1,w2,w3	: New weights.

Scheduling	: This functions performs weighted round robin or weighted fair queuing scheduling.

Attributes :

simulation_time	: This is the time quantum used for simulation.

SIMULATION RESULTS AND DISCUSSION

The curves show the effect of adaptive performance management.

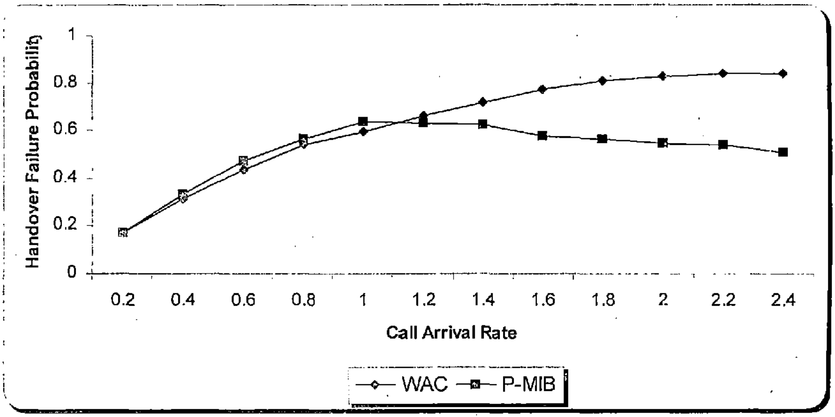


Fig 5.1 Effect of adaptive performance management on handover failure probability.

Fig. 5.1 plots the handover failure probability with and without adaptive performance management. Curves with adaptive performance management are denoted with MIB and WAC denotes curve with out adaptive control. From the curve it is clear that with adaptive performance the handover failure probability can be kept low. This is due to the fact that the handover bandwidth is updated according to handover failure probability. As from Fig. 5.3, the adjusted bandwidth for low arrival rate i.e., 1 arrival per second is kept constant which is equal to 7%. As the arrival rate increases the portion of handover bandwidth increases that's why the handover failure probability increases first and then decreases. The handover failure probability is calculated as follows

Handoff failure probability = $b_h^{(new)}$/(call blocking probability)* $b_h^{(old)}$

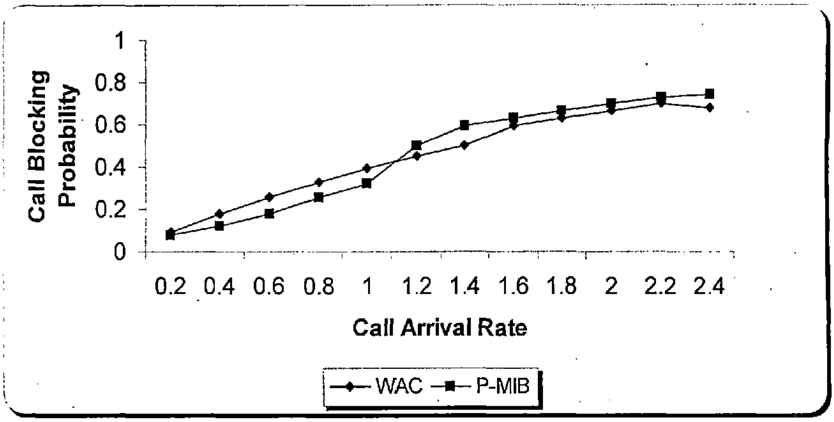


Fig 5.2 Effect of adaptive performance management on new call blocking probability.

Fig. 5.2 shows the new call blocking probability, as from the curve the call blocking probability for adaptive control is higher as compared to non-adaptive control. This is due to the fact that for low arrival rate the handover bandwidth is kept low but as the arrival rate is increased the bandwidth is also increased for handover, which in turn increases the blocking probability.

call blocking probability $=b_h^{(new)}/ b_h^{(old)} *$ (handoff failure probability)

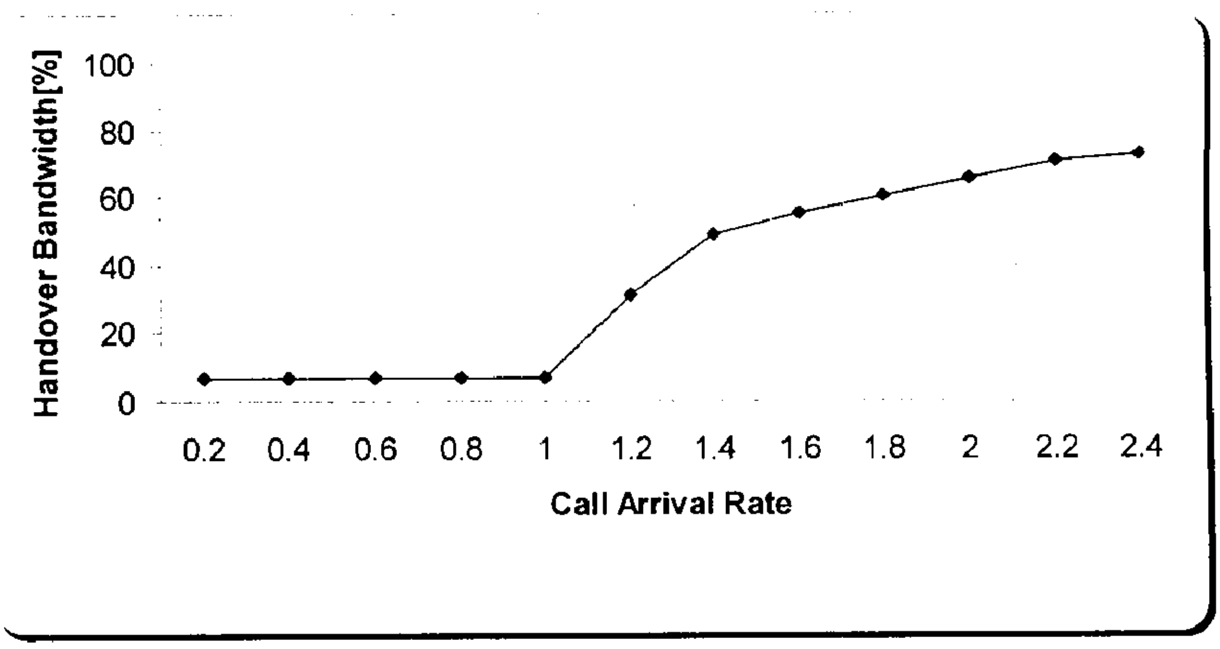


Fig 5.3 Adaptive adjustment of handover bandwidth.

Fig. 5.3 shows the adaptive adjustment of handover bandwidth. The handover bandwidth is adjusted according to handover failure probability and new call blocking probability. Whenever the handover failure probability increases the handover bandwidth is increased as follows.

For low arrival rate i.e., up to 1arrival per second the handover is kept to equal 7% of the over all bandwidth. The handover bandwidth is updated according to given formula i.e.,

$b_h^{(new)}$=(handoff failure probability)*(call blocking probability)* $b_h^{(old)}$

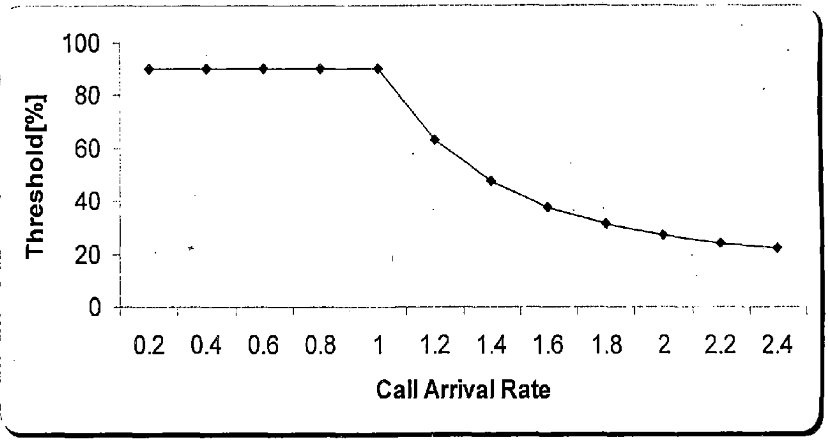


Fig 5.4 Adaptive adjustment of NRTQ threshold.

Fig. 5.4 Show the adjustment of NRTQ threshold. The NRTQ threshold is adjusted according to

new threshold = (packet loss probability * old threshold)

The NRTQ threshold is kept constant to 90% for arrival rate below 1 arrivals per time unit. And for high arrival rate the threshold is decreased. Therefore the packet loss probability increases compared to the case without adaptive control, which is shown in Fig. 5.5.

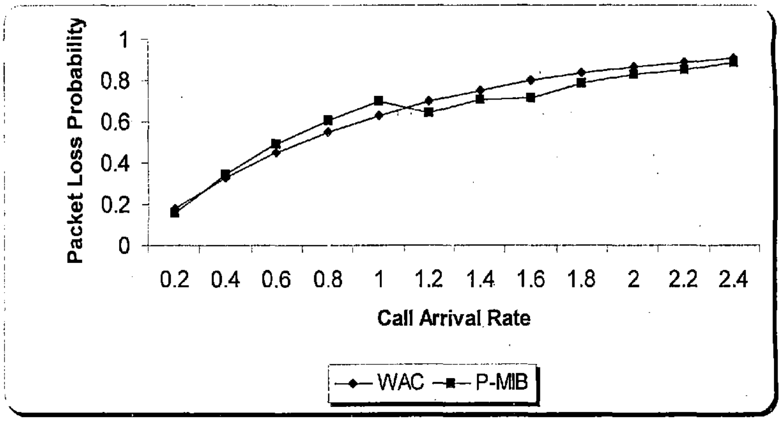


Fig 5.5 Effect of adaptive performance management on packet loss probability.

Fig. 5.5 shows a significant improvement for the packet loss probability due to adaptively adjusting the threshold of non-real-time queue threshold.

packet loss probability = (total cells lost)/total cell generated. Adaptive packet loss probability is calculated according to

new threshold = packet loss probability*old threshold.

Packet loss probability = (new threshold)/old threshold.

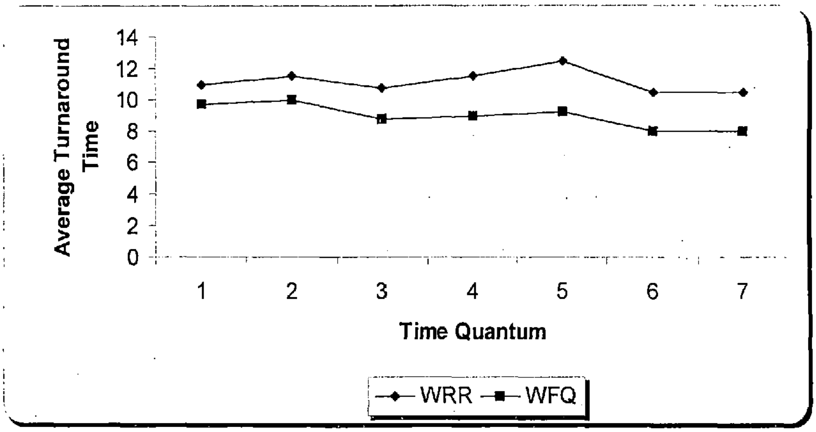


Fig 5.6 The average turnaround time at different time quantum for variable packet size.

Fig. 5.6 represents the average turnaround time for different time quantum's. Turnaround time is the time interval from the submission of the packet to the time of completion of the transmission. From the graph it is clear that the average turnaround time for weighted round robin scheduling is high as compared to weighted fair queuing scheduling with variable packet size.

At different time quantum

Avg. trn time =(sum of time taken by each packet to transmit)/number of packets

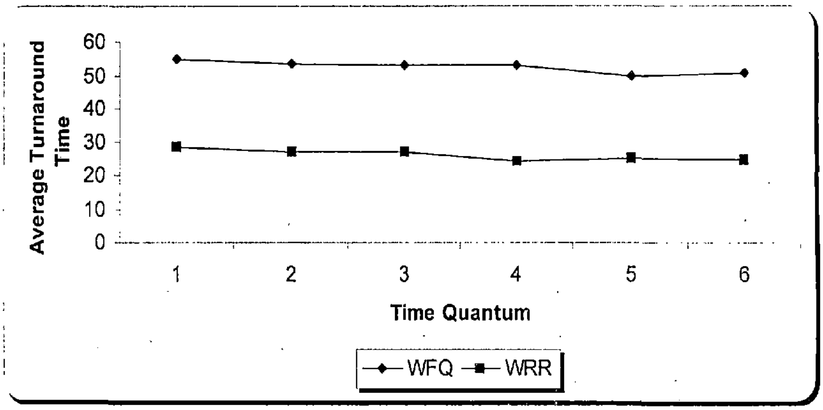


Fig 5.7 The average turnaround time at different time quantum for fixed packet size.

Fig. 5.7 represents the average turnaround time for different time quantum, using two scheduling scheme i.e., weighted round robin and weighted fair queuing. From the graph it is clear that the average turnaround time for weighed round robin scheduling is low with respect to weighed fair queuing at different time quantum. The packets are of fixed size.

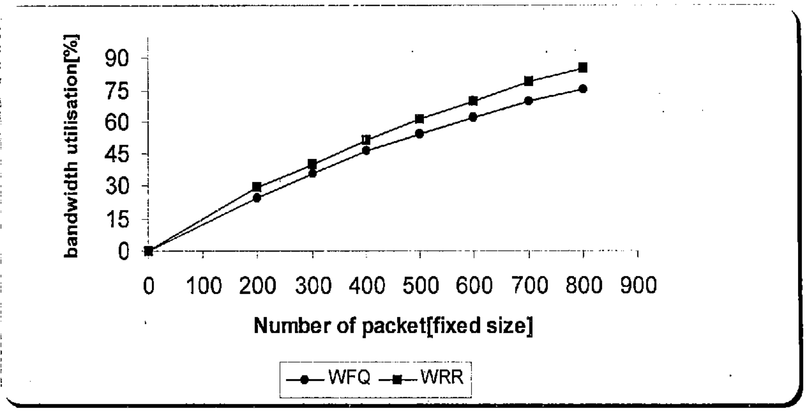


Fig 5.8 bandwidth utilization for fixed packet size.

Fig. 5.8 represent bandwidth utilization of the system by using different scheduling scheme. The bandwidth utilization is calculated for fixed packet size. As from Fig. 5.7 the average turnaround time for weighted round robin is less than the average turnaround time of weighted fair queuing scheduling, which means that more number of packet's can be served in less time. Hence the bandwidth utilization increases for weighted round robin scheduling.

Bandwidth utilization is calculated as follows

$$\text{Bandwidth utilization} = \frac{S_b}{t_b} * 100$$

S_b=sum of bandwidth for each packet

t_b=total bandwidth*100

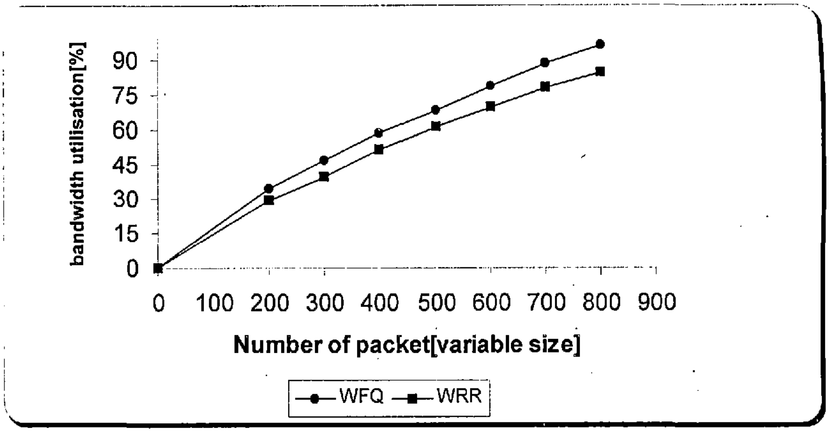


Fig 5.9 bandwidth utilization for variable packet size.

Fig. 5.9 represents the bandwidth utilization of the system for variable size packet. As the average turnaround time for weighted fair queuing is less than the average turnaround time of weighted round robin scheduling. Hence the bandwidth utilization in case of weighted fair queuing is much better then the weighted round robin scheduling for variable size packets.

To derive variable size packets lognormal distribution is used i.e.,

Size of packet = $\frac{1}{x\sqrt{2\pi\sigma^2}}\exp-\frac{-(\ln x-\mu)^2}{2\sigma^2}$

Where x>0

μ=1.8821 and σ^2=5.4139 are fixed for each packet.

Bandwidth utilization = $\frac{s_b}{t_b}$*100

S_b=sum of bandwidth for each packet

t_b=total bandwidth*100

CONCLUSIONS

6.1 Conclusions

In this dissertation P-MIB has been implemented for dynamically adjusting the packet scheduler and admission controller of the Universal Mobile Telecommunications System. The aim of this adaptive control framework lies in improving bandwidth utilization of the Universal Mobile Telecommunications System radio channels. This framework distinguishes three different type of quality of services: circuit-switched services as well as packet-switched RT service and NRT services. The P-MIB adaptively adjusts the system parameters of the admission controller.

Controlled system parameters constitute the portion of bandwidth reserved for handovers, the buffer threshold of NRTQ, and the queuing weights for scheduling NRT packets by packet scheduler. The performance curves for various QoS measures to illustrate the benefit of the P-MIB have been shown in chapter 5. From the curves it is clear that the adaptive performance management achieves a significant improvement handover failure probability and packet loss probability. The weighted round robin packet scheduling is better for fixed packet size while the weighted fair queuing is better to use in case if variable size packets are considered.

6.2 Scope for further work

- In this dissertation the adaptive control framework is tailored for UMTS networks and also the services and QoS profiles standardized for UMTS is being considered. However considering other services and QoS profile, the basic idea underlying the control framework can also be applied for adaptive control of wire-line networks.

REFERENCES

[1]. Christoph Lindemann, Macro Lohmann, Axel Thummler. "Adaptive performance management for universal mobile telecommunication system networks", *computer networks* 38 (2002) 477-496.

[2]. Mitrevski Kivil, Zgonjanin Dus Ko." Principle of the UMTS terrestrial radio access networks-UTRAN", Telecommunication forum telfor'2001, Beograd.

[3]. J.F. Huber, D. Weiler, H. Brand," UMTS, the mobile multimedia vision for IMT-2000: a focus on standardization", *IEEE communication magazine* 38(2000) 129-136.

[4]. Remco Litjens, "The Impact Of Mobility On UMTS Network Planning", *computer networks* 38 (2002) 497-515.

[5]. E. A nderlind. J. Zander, " A traffic model for non real-time data users in a wireless radio network", *IEEE Communication Letters* 1(1997) 37-39.

[6]. 3GPP, *http://www.3gpp.org.*

[7]. Javier Zamore, Stephen, Alexandros, Eleftheriadis, Shih-Fu Chang and Dimitris," A Practical Methodology For Guaranteeing Quality Of Service For Video-on-Demand", *IEEE Transactions On Circuits and system For Video Technology.* Vol. 10, No.1. (2000).

[8]. D. Bertsekas and R.Gallager, Data Networks, Prentice-Hall Of India Pvt. Ltd. 1999.

[9]. H. Zhang, "Service disciplines for guaranteed performance service in packet-switched networks", *Proceeding of the IEEE* 83 (1995) 1374-1396.

[[10]. UMTS-Forum, UMTS-2000 spectrum, Report No. 6 1999, *http://www.etsi.org/.*

[11]. T. Ojanpera, R. Prasad, " An overview of air interface multiple access for IMT-2000/UMTS", *IEEE Communications Magazine.* 36 (9) (1998) 82-95.

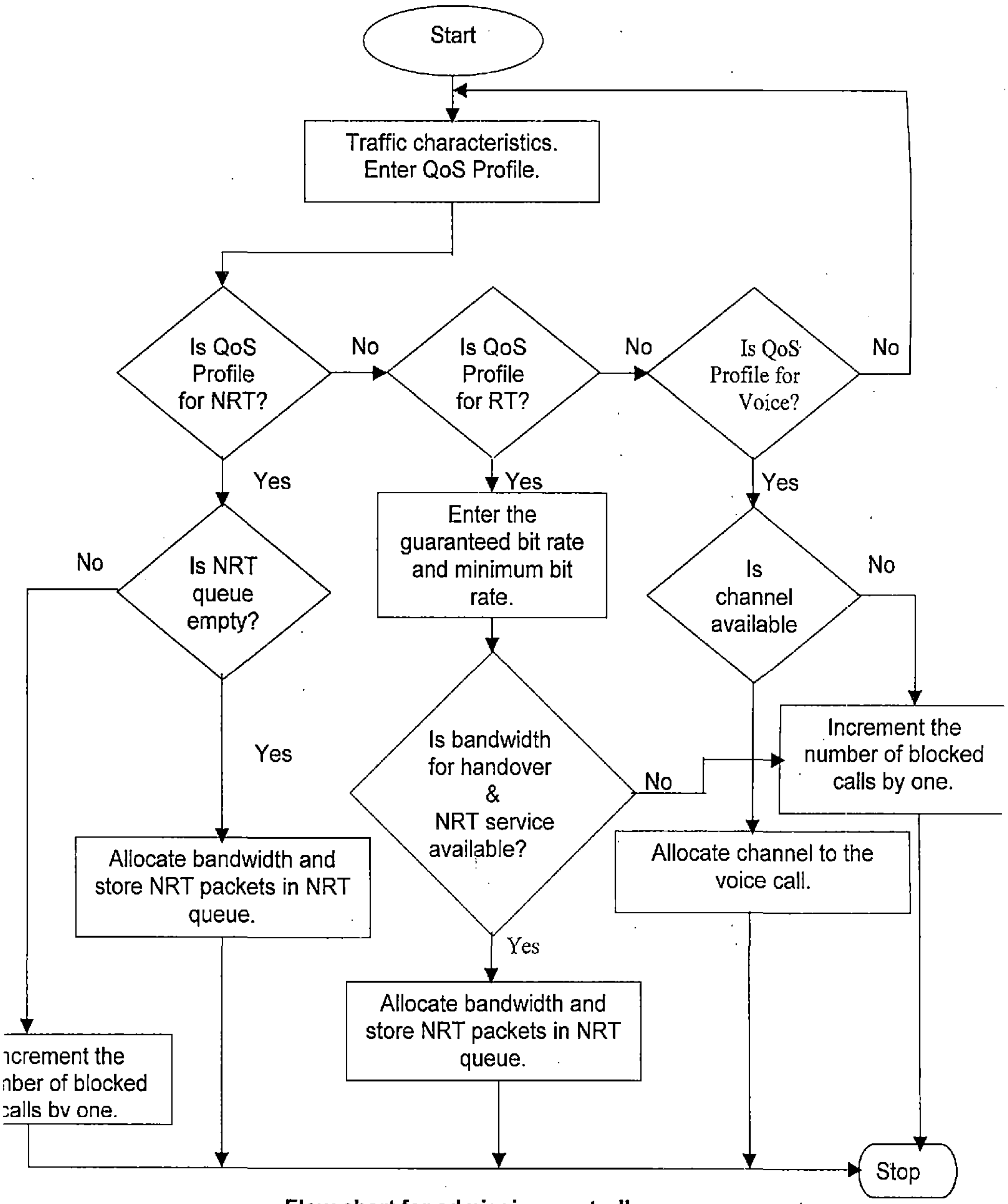
[12]. Andrew S. Tenenbaum, computer networks, Third Edition, Prentice-Hall of India Pvt. Ltd. 1998.

[13]. Jochen Schiller, Mobile Communications, Addison Wesley Longman,

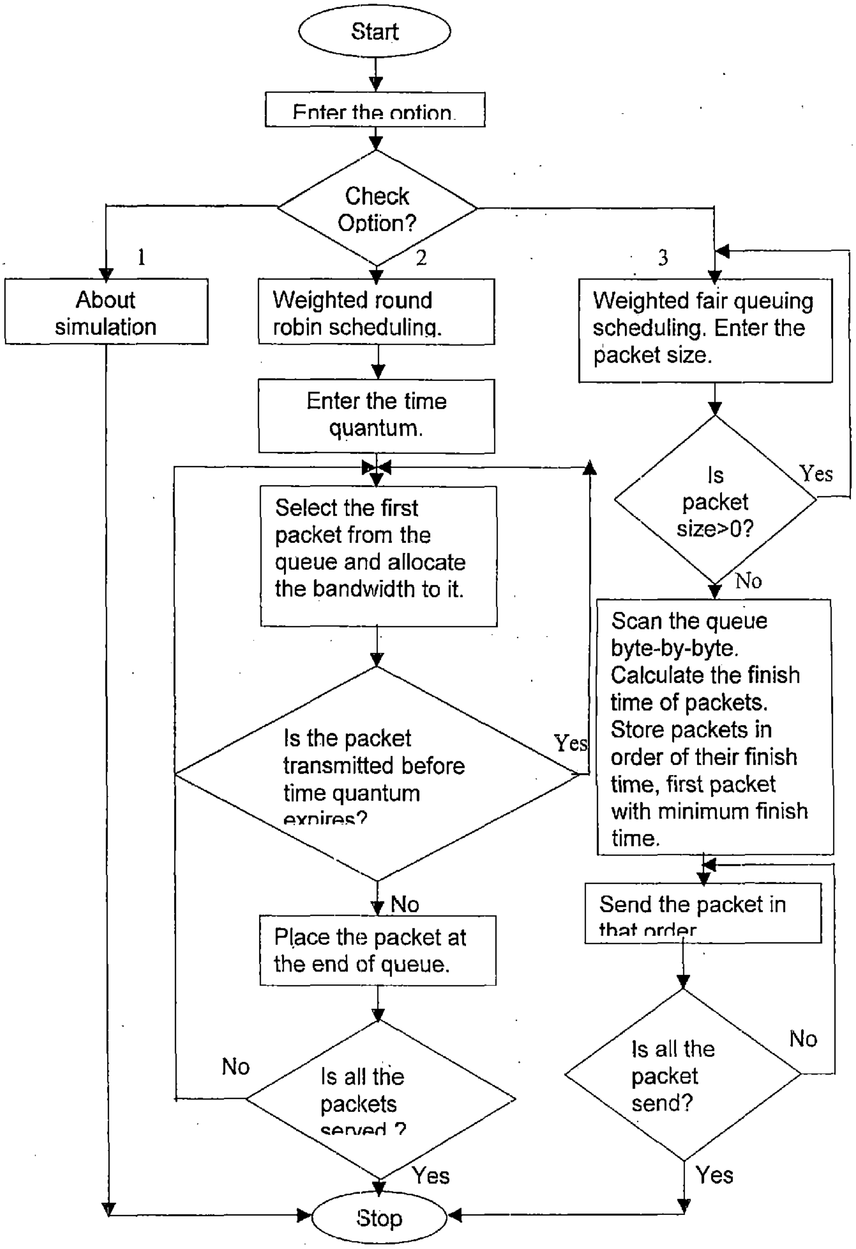
Pearson Education Ltd. 2000.

[14]. V. Bharghavan, S. Lu, T. Nandagopal, " Fair queuing in wireless networks: issue and approaches", *IEEE Personal Communications* 6 (1999) 44-53.

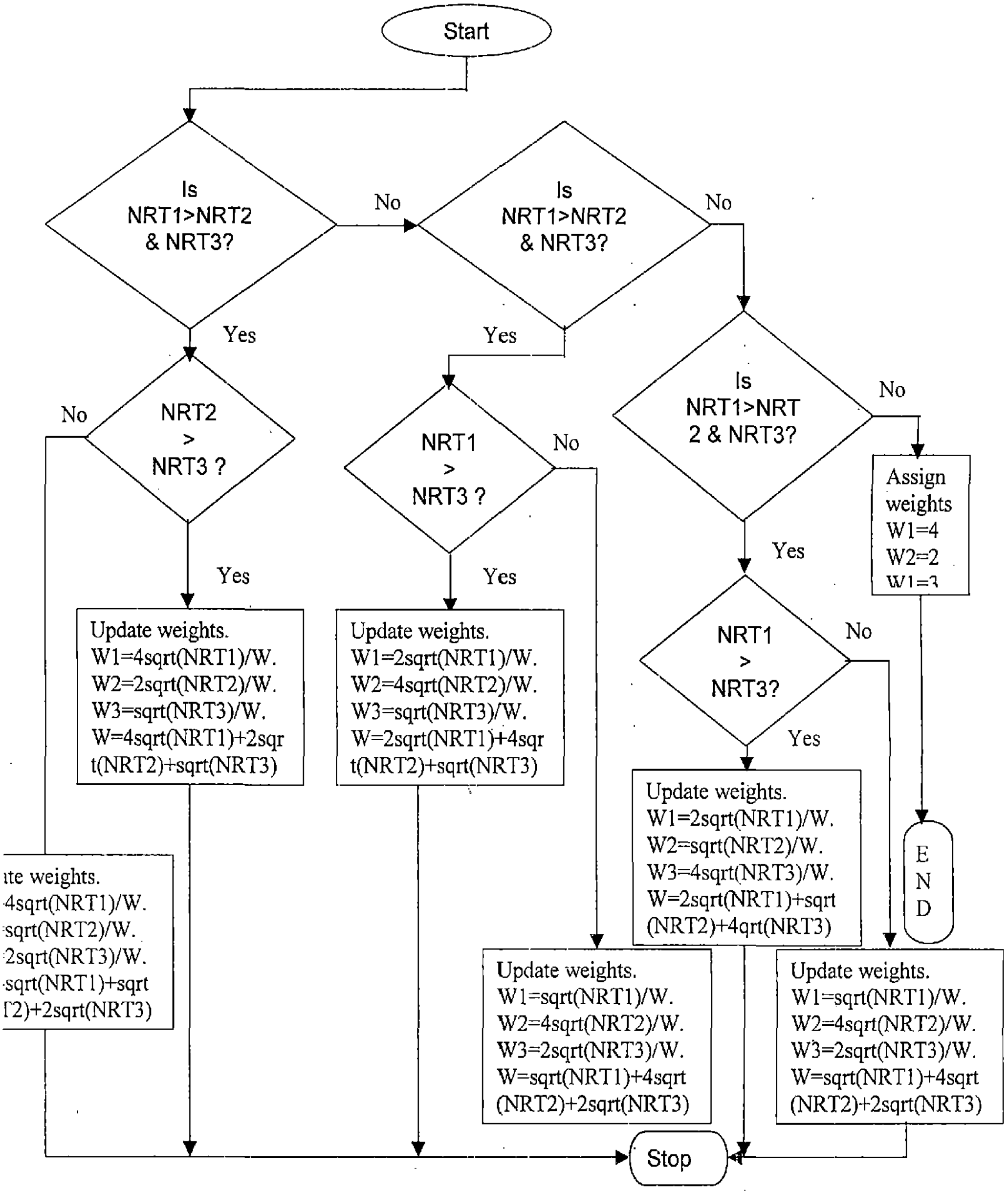
[15]. Nilsson, " Toward third-generation wireless communication", Ericsson Review No.2, *http://www.ericsson.com/.*


Start
Traffic characteristics. Enter QoS Profile.
Is QoS Profile for NRT?
No
Is QoS Profile for RT?
No
Is QoS Profile for Voice?
No
Yes
Yes
Yes
Is NRT queue empty?
No
Yes
Enter the guaranteed bit rate and minimum bit rate.
Is channel available
No
Is bandwidth for handover & NRT service available?
No
Yes
Increment the number of blocked calls by one.
Allocate bandwidth and store NRT packets in NRT queue.
Allocate channel to the voice call.
Allocate bandwidth and store NRT packets in NRT queue.
ıcrement the ıber of blocked ɔalls bv one.
Stop

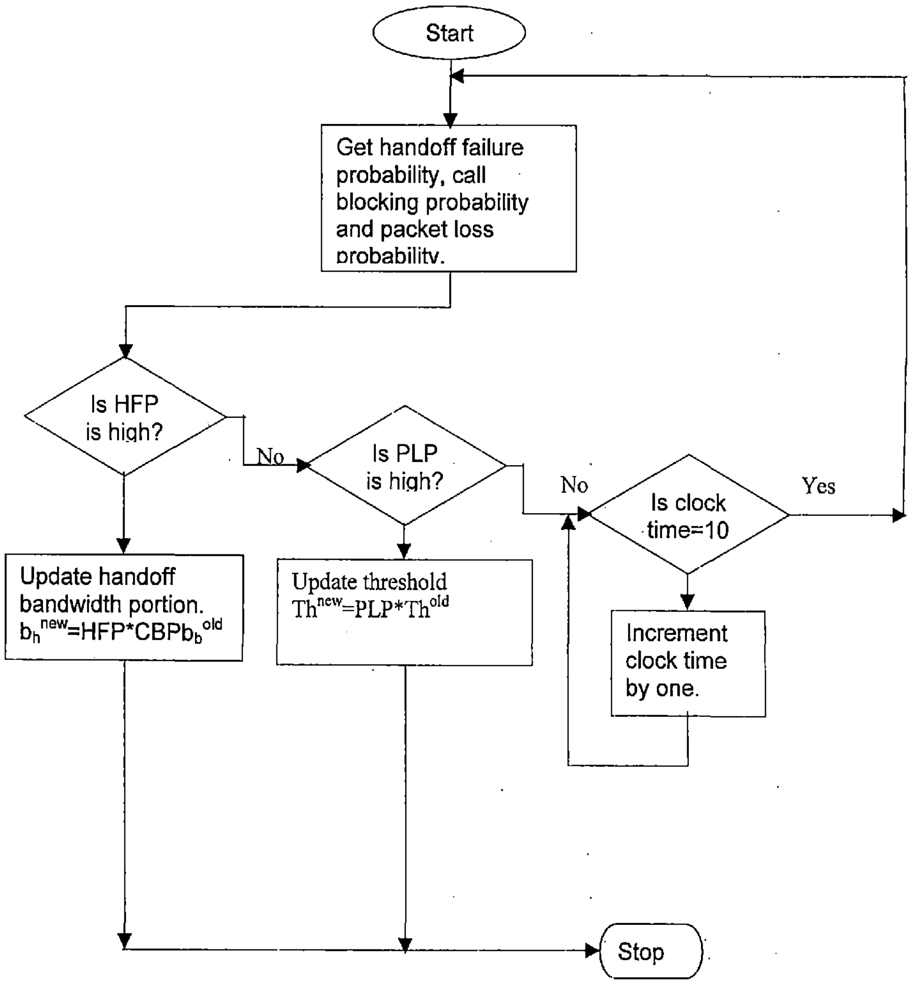
Flow chart for admission controller


Start
Enter the option.
Check Option?
1
2
3
About simulation
Weighted round robin scheduling.
Weighted fair queuing scheduling. Enter the packet size.
Enter the time quantum.
Is packet size>0?
Yes
No
Select the first packet from the queue and allocate the bandwidth to it.
Scan the queue byte-by-byte. Calculate the finish time of packets. Store packets in order of their finish time, first packet with minimum finish time.
Is the packet transmitted before time quantum expires?
Yes
No
Place the packet at the end of queue.
Send the packet in that order
Is all the packets served ?
No
Yes
Is all the packet send?
No
Yes
Stop

Flow chart for packet scheduler


Start
Is NRT1>NRT2 & NRT3?
No
Yes
Is NRT1>NRT2 & NRT3?
No
Yes
Is NRT1>NRT 2 & NRT3?
No
Yes
NRT2 > NRT3 ?
No
Yes
NRT1 > NRT3 ?
No
Yes
NRT1 > NRT3?
No
Yes
Assign weights W1=4 W2=2 W1=3
Update weights. W1=4sqrt(NRT1)/W. W2=2sqrt(NRT2)/W. W3=sqrt(NRT3)/W. W=4sqrt(NRT1)+2sqrt(NRT2)+sqrt(NRT3)
Update weights. W1=2sqrt(NRT1)/W. W2=4sqrt(NRT2)/W. W3=sqrt(NRT3)/W. W=2sqrt(NRT1)+4sqrt(NRT2)+sqrt(NRT3)
Update weights. W1=2sqrt(NRT1)/W. W2=sqrt(NRT2)/W. W3=4sqrt(NRT3)/W. W=2sqrt(NRT1)+sqrt (NRT2)+4qrt(NRT3)
ite weights. 4sqrt(NRT1)/W. sqrt(NRT2)/W. 2sqrt(NRT3)/W. sqrt(NRT1)+sqrt [2)+2sqrt(NRT3)
Update weights. W1=sqrt(NRT1)/W. W2=4sqrt(NRT2)/W. W3=2sqrt(NRT3)/W. W=sqrt(NRT1)+4sqrt (NRT2)+2sqrt(NRT3)
Update weights. W1=sqrt(NRT1)/W. W2=4sqrt(NRT2)/W. W3=2sqrt(NRT3)/W. W=sqrt(NRT1)+4sqrt (NRT2)+2sqrt(NRT3)
E N D
Stop

Flow chart for controlling weights


Start
Get handoff failure probability, call blocking probability and packet loss probability.
Is HFP is high?
No
Is PLP is high?
No
Is clock time=10
Yes
Update handoff bandwidth portion. b_h^{new}=HFP*CBPb_b^{old}
Update threshold Th^{new}=PLP*Th^{old}
Increment clock time by one.
Stop

Flow chart for adjusting handover bandwidth and NRT threshold

APPENDIX- B

Software Listing

```
#include<iostream.h>
#include<stdlib.h>
#include<string.h>
#include<stdio.h>
#include<process.h>
#include<conio.h>
#include<dos.h>
#include<graphics.h>
#include<string.h>
#include<math.h>
#include<fstream.h>

#include"d:\tc\bin\devs\queue.h"
#include"d:\tc\bin\devs\rand.h"
#include"d:\tc\bin\devs\sml.h"

#define AVAILABLE_BANDWIDTH 7680
#define RTQ_BUFFER 1000
#define NRTQ_BUFFER 1000
#define THRESHOLD 0.9
#define FCFS 0.65
#define HANDOVER_BANDWIDTH 0.5
#define VOICECALL_BANDWIDTH 0.1
#define DATAPACKET_BANDWIDTH 0.2
#define D 1
#define DRAW 1
#define DELETE 2


int WinLeft=5, WinTop=3, WinDepth=20, WinRight=75;
int flag=0;
void scroll(int x,int y,int,char *str,char,char);
void display1(void);
int StructLen;
unsigned char buffer1[4000],buffer2[4000];
char*fileName;
FILE *source_record;
int c=240;
int k;
//////////////////////////////////////////////////////////////////////////////////////////////////////////
struct MESSAGE{
                char text[200];
                }message;

//////////////////////////////////////////////////////////////////////////////////////////////////////////

typedef struct Menu
   {
    int no;
    unsigned char item[40];
    unsigned char help[50];
   };

//////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
struct
        {
         int no;
         unsigned char name[10];
         unsigned char help[50];
         int xcord;
         int ycord;
         }  MainMenu[4]=
           {
                {1,"CON ADM","1: NETWORK CONFIGURATION ",2,3},
                {2,"TUT ADM","2: HELP TUTORIAL ",17,3},
                {3,"SIM ADM","3: START SIMULATION",35,3},
                {4,"CHECK ADM","4: CHECK RESULTS",45,3}
           };
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
struct Menu  ConfigMenu[]=
        {
         {1,"1: MOBILE SESSION ","ENTER 1 FOR TRAFFIC TYPE"},
        };
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
struct Menu  TutMenu[]=
        {
         {1,"1: ABOUT UMTS ","1 FOR UMTS INTRO"},
         {2,"2: ABOUT UMTS ARCHITECTURE","2 FOR ARCHITECTURE INFORMATION"},
         {3,"3: ABOUT QoS OF UMTS","3 FOR QoS "},
         {4,"4: ABOUT ADMISSION CONTROL","4 FOR ADMISSION CONTROL "},
         {99,"5: GRAPHICAL REPRESENTATION ","5 FOR DEMO STEPS"}
         };
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
struct Menu SimuMenu[]=
        {
         {1,"1: ABOUT SCHEDULING ",""},
         {2,"2: WEIGHTED ROUND ROBIN  ",""},
         {99,"3: WEIGHTED FAIR QUEUING ",""}
        };
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
struct Menu CheckMenu[]=
        {
         {1,"1: WEIGHTED ROUND ROBIN FILE ",""},
         {2,"2: WEIGHTED FAIR QUEUING FILE",""},
        };
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
struct Menu Heading[]=
                {
                 {1,"1: NETWORK CONFIGURATION MENU","1:INITIALIZATION "},
                 {2,"2: TUTORIAL MENU","2:INFORMATION AND GUIDELINES"},
                 {3,"3: SCHEDULING & SIMULATION MENU","3:STARTING THE
                  SIMULATION"},
                 {99,"4: CHECK RESULTS MENU","4:READ THE ASSOCIATED OUTPUT"}
                 };


//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
int DoMainChoice(int choice);
void Beep(void );
void DisplayHelpBox(void );
void DisplayMain(void );
void DrawBar(int ,int ,int ,int ,int);
void DisplayMenu(int c ,int n, struct Menu *temp);
int GetStructLen(struct Menu *TempStruct);
int GetChoice(int);
void Mobile_subchoice(int);
void DoSubChoice(int ch);
void DoSubChoice1(int ch);
void DoSubChoice2(int Choice);
void DoSubChoice3(int ch);
void ConfigAdmin(void ) ;
void TutAdmin(void ) ;
void CheckAdmin(void ) ;
void Scheduling();
void disp();
void vadmission_check(float);
void rtadmission_check(int);
void ntadmission_check(int);
void byte_scan();
void p_mib();
void fileread(FILE *);

////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int main(void){
clrscr();
int choice;
    int i;
    clrscr();
    DisplayHelpBox();
    DisplayMain();
    DisplayMenu(4,1,Heading);
    StructLen=GetStructLen(Heading);
    choice = GetChoice(StructLen);
    DrawBar(WinLeft,WinTop,WinRight,WinDepth,DELETE);
    DoMainChoice(choice);
    return(0);
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void DisplayHelpBox(void )
 {
 window(5,22,75,24);
 textbackground(GREEN);
 clrscr();

  }

////////////////////////////////////////////////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
void DisplayMain(void )
 {
   int i;
   static int j =1;
   if(j==1)
   {
   display1();
   j=j+1;
   }
   DrawBar(WinLeft,WinTop,WinRight,WinDepth,DRAW);
   gotoxy(23,2);
   cputs("     UMTS MAIN MENU ");
   gotoxy(23,3);
   cputs("-----------------------");
   for(i=0;i<4;i++)
   {
    gotoxy(WinLeft+10,5+i);
    cputs(MainMenu[i].help);
   }
   gotoxy(WinLeft+10,WinTop+8);
   cputs(" ENTER YOUR CHOICE,0 EXIT :");
   }

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void DisplayMenu(int index,int Number,struct Menu MemberMenu[])
 {
  int i,Lenth;
  clrscr();
  gettext(1,1,80,25,buffer1);
  DisplayHelpBox( );
  for(i=0;i<index;i++)
  cputs(MemberMenu[i].help);
  DrawBar(WinLeft,WinTop,WinRight,WinDepth,DRAW);
  gotoxy(23,2);
  cputs(" UMTS MAIN MENU             ");
  Lenth=strlen(Heading[Number].item);
  for(i=0;i<Lenth;i++)
  {
   gotoxy(23+i,3);
   cputs("-");
  }
  for(i=0;i<index;i++)
  {
  gotoxy(WinLeft+10,5+i);
  puts(MemberMenu[i].item);
  }
  gotoxy(WinLeft+12,WinTop+i+4);
  cputs("ENTER YOUR CHOICE ,0 TO EXIT: ");
 }

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
int GetStructLen(struct Menu TempStruct[])
{
int i=0;
for(i=0;i<99;i++)
{
```

```
if(TempStruct[i].no==99)
break ;
}
return(i);
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////
int GetChoice(int Len)
{
 int Choice;
 Choice= (int)getche();
if((Choice-48)>9||(Choice-48)<0)
{
Beep();
}
  gotoxy(WinLeft+10,7+Len);
  cputs("                    ");
 return(Choice-48);
 }
////////////////////////////////////////////////////////////////////////////////////////////////////////////
 void Beep(void )
{
 sound(3000);
 delay(100);
 nosound();
 gotoxy(WinLeft,WinDepth-3);
 cputs("WRONG ENTRY TRY AGAIN");
 delay(100);
 gotoxy(WinLeft,WinDepth-3);
 cputs("                    ");

 }
////////////////////////////////////////////////////////////////////////////////////////////////////////////
int DoMainChoice(int Choice)
 {
  int i;
  switch(Choice)
  {
    case 1:
            ConfigAdmin();
            break;

   case 2:
             TutAdmin();
             break;

   case 3:
            Scheduling();
            break;

   case 4:
            CheckAdmin();
            gotoxy(5,5);
            main();
            break;

   case 0:
```

```
            exit(0);

  default:
            main();
  }
 return(0);
}
/////////////////////////////////////////////////////////////////////////////////////////////////////////
void CheckAdmin(void )
  {
  int i ,SubChoice;
  clrscr();
  DisplayMenu(2,1,CheckMenu);
  StructLen=GetStructLen(SimuMenu);
  SubChoice= GetChoice(StructLen);
  DrawBar(WinLeft,WinTop,WinRight,WinDepth,DELETE);
  DoSubChoice3(SubChoice);
  }
/////////////////////////////////////////////////////////////////////////////////////////////////////////
void DoSubChoice3(int Choice)
{
 int i;
 int j;
 switch(Choice)
 {
  case 1:
            {
             DisplayHelpBox( );
             cprintf("  \n out put are ..");
             DrawBar(WinLeft,WinTop,WinRight,WinDepth,DRAW);
             clrscr();
             int i;
             gotoxy(2,2);
             cprintf("\n\rClearing the source log file");
             for(i=0;i<5;i++)
             {
              cprintf(".");
              delay(200);
             }
              remove("wrrobin");
              source_record=fopen("wrrobin.txt","a+");
              fclose(source_record);
              source_record = fopen("wrrobin.txt","r");
              fileread(source_record);
              fwrite(&message,strlen(message.text),1,source_record);
              fclose(source_record);
              getch();

              main();
              break;
            }
case 2:
            {
             DisplayHelpBox( );
             cprintf("  \n out put are ..");
             DrawBar(WinLeft,WinTop,WinRight,WinDepth,DRAW);
            clrscr();
            int i;
            gotoxy(2,2);
            cprintf("\n\rClearing the source log file");
            for(i=0;i<5;i++)
            {
             cprintf(".");
             delay(200);
            }
             remove("wfqueuing");
             source_record=fopen("wfqueuing.txt","a+");
             fclose(source_record);
             source_record = fopen("wfqueuing.txt","r");
             fileread(source_record);
             fwrite(&message,strlen(message.text),1,source_record);
             fclose(source_record);
             getch();

             main();
             break;
            }

 default:
            main();
 }
}
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
 void fileread(FILE *fp)
{
char ch;
int i,j,x;
char str[2];
clrscr();
gotoxy(1,1);
textbackground(YELLOW);
textcolor(RED);
textbackground(2);
textcolor(RED);
ch=fgetc(fp);
i=2,j=1,x=0;
while(ch!=EOF)
{
int a;
delay(10);
str[0]=ch;
str[1]='\0';
if(str[0]=='\n')
{
printf("\n");
j++;
i=2;
x++;
}
else
{
gotoxy(i,j);
```

```
cprintf("%s",str);
}

if(x==16)
{
 x=0;
 getch();
 clrscr();
 j=1;
}
ch=fgetc(fp);
i++;
}
getch();

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void Scheduling()
{
  char ch;
  int i,c=240 ,SubChoice;
  clrscr();
  DisplayMenu(3,1,SimuMenu);
  StructLen=GetStructLen(SimuMenu);
  SubChoice= GetChoice(StructLen);
  DisplayHelpBox();
  DrawBar(WinLeft,WinTop,WinRight,WinDepth,DELETE);
  DoSubChoice2(SubChoice);
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void DoSubChoice2(int Choice)
{
 int i;
 switch(Choice)
 {
  case 1:  {

            DrawBar(WinLeft,WinTop,WinRight,WinDepth,DRAW);
            gotoxy(5,5);
            cputs("At radio network controller responsible for a cell");
            gotoxy(5,6);
            cputs("cluster, data packets from various connection arrive ");
            gotoxy(5,7);
            cputs("and are queued until bandwidth for transmission is available");
            gotoxy(5,8);
            cputs("service discipline at this base station controls the order ");
            gotoxy(5,9);
            cputs("in which packets are served and how packets in transfer share ");
            gotoxy(5,10);
            cputs("the available bandwidth. ");
            gotoxy(5,12);
            cputs("scheduling policies are.");
            gotoxy(5,13);
```

```
          cputs("WEIGHTED ROUND ROBIN ");
          gotoxy(5,14);
          cputs("WEIGHTED FAIR QUEUING");
          getch();
          main();
          break;
          }
case 2:
          {
          int simulation_time;
          int avg_turnaround_time,num_of_packets;
         DisplayHelpBox();
         cprintf("HELP : \n Simulator will take this entered time as the initial time..");
         cprintf(" If this option is not visited Simulator will take initial time to be zero");
         DrawBar(WinLeft,WinTop,WinRight,WinDepth,DRAW);
         gotoxy(5,5);
         cputs("WEIGHTED ROUND ROBIN SCHEME IS SELECTED");
         gotoxy(5,7);
         cputs("Enter the time slice:");
         scanf("%d",&simulation_time);
         gotoxy(5,8);
                 cputs(" Radio channels to be Reserved: ");
                 scanf("%d",&resRadio);
                 gotoxy(5,9);
                 cputs("Save Results on File ");
                 cin>>fileName;
                 gotoxy(5,10);
                 cputs("Enter the number of packets");
                 scanf("%d",&num_of_packets);
                 simulation Rcell(WRR,fileName);
                 clrscr();
                 cputs(" Simulation for WEIGHTED ROUND ROBIN scheme ");
                 gotoxy(5,5);
                 cputs(" SIMULATING ");
                 gotoxy(5,8);
                           for( i=1;i<26;i++){
                            Rcell.traffic(c,50);
                            Rcell.start();
                            Rcell.get_packet_size(num_of_packets);
                            Rcell. bandwidth_utilisation(int numofpkt);

                            Rcell.save();
                            c+=120;
                            cputs("Þ³");
                            delay(500);
                           }
                           Rcell.p_mib();
                 gotoxy(5,12);
                 cputs("END OF SIMULATION ");
                 gotoxy(5,13);
                 cputs("Press any key...");
        getch();
         main();
         break;
         }
```

```
 case 3:
         {
          int i,num_of_packets;
          DrawBar(WinLeft,WinTop,WinRight,WinDepth,DRAW);
          gotoxy(5,5);


cputs("WEIGHTED FAIR QUEUING IS SELECTED");
          gotoxy(5,6);
          cputs("BYTE_BY_BYTE SCANING OF QUEUES IS GONIG ON...");
           delay(1500);
          byte_scan();

          gotoxy(5,8);
                  cputs(" Radio channels to be Reserved: ");
                  scanf("%d",&resRadio);
          gotoxy(5,9);
          cputs("Save Results on File ");
          cin>>fileName;
          gotoxy(5,10);
                  cputs("Enter the number of packets");
                  scanf("%d",&num_of_packets);


          simulation Mcell(WFQ,fileName);
          clrscr();
                  cputs("Simulation for WEIGHTED FAIR QUEUING scheme ");
                  gotoxy(5,4);
                  cputs("SIMULATING ");
                  gotoxy(5,8);

                           for( i=1;i<30;i++){
                            Mcell.traffic(c,50);
                            Mcell.start();
                            Mcell.  get_packet_size(num_of_packets);
                            Mcell. bandwidth_utilisation(int numofpkt);

                            Mcell.save();
                            c+=120;
                            cputs("Þ³");
                            delay(500);

                            }
                  Mcell.p_mib();
                  gotoxy(5,12);
                  cputs("END OF SIMULATION ");
                  gotoxy(5,13);
                  cputs("Press any key...");
                  getch();
          main();
          break;
          }
 default: main();
 }

}
```

```
for(i=0;i<50;i++)
{
NEW_HANDOVER_BANDWIDTH[i]=HANDOFF_FAIL_PROB[i]*CALL_BLOCK_PROB[i]*HAND
OVER_BANDWIDTH;
NEW_THRESHOLD[i]=PKT_LOSS_PROB[i]*THRESHOLD;
}
int NRT1=400;
int NRT2=200;
int NRT3=100;
 float w=4*sqrt(NRT1)+2*sqrt(NRT2)+sqrt(NRT3);
 float w1=4*sqrt(NRT1)/w;
 float w2=2*sqrt(NRT2)/w;
 float w3=sqrt(NRT3)/w;

}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void byte_scan()
{
 int arr[25];
 int static count=0,finish_time[20],np;
 for(int i=1;i<=np;i++)
 for(int j=1;j<=8;j++)
 finish_time[i]=count++;

int temp;
for(i=0;i<=np;i++)
{
if(arr[i+1]<arr[i])
{
temp=arr[i+1];
arr[i+1]=arr[i];
arr[i]=temp;
}
else
arr[i];
}

}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void ConfigAdmin(void )
  {
  int i ,SubChoice;
  clrscr();
  gettext(1,1,80,25,buffer1);
  DisplayMenu(1,1,ConfigMenu);
  StructLen=GetStructLen(ConfigMenu);
  SubChoice= GetChoice(StructLen);
  DrawBar(WinLeft,WinTop,WinRight,WinDepth,DELETE);
  DoSubChoice(SubChoice);
  }
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
void DoSubChoice(int Choice)
{
 int i;
 int j;
 switch(Choice)
 {
  case 1:
            {
             int get;
             DisplayHelpBox( );
             cprintf("HELP : \n Please enter the traffic type.");
             DrawBar(WinLeft,WinTop,WinRight,WinDepth,DRAW);
             gotoxy(30,3);
             cout<<"MOBILE SESSION\n";
             gotoxy(5,6);
             cout<<"1. Voice Data\t";
             cout<<"2. Real Time Data\t";
             cout<<"3. Non Real Time Data";
             gotoxy(5,8);
             cout<<"Enter your choice:";
             cin>>get;
             Mobile_subchoice(get);

             getch();
             main();
             break;
            }
   default:  main();
  }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void Mobile_subchoice(int c)
{
switch(c)

          {
case 1:
          int vbwidth;
          DrawBar(WinLeft,WinTop,WinRight,WinDepth,DRAW);
          gotoxy(5,3);
          cout<<"voice data is selected......";
          gotoxy(5,5);
          cout<<"Enter the required bandwidth\t";
          cin>>vbwidth;
          gotoxy(5,7);

          cout<<"Admission control checking for "<<vbwidth<<"  kbps bandwidth......";
          vadmission_check(vbwidth);
          getch();
          main();
          break;
case 2:
          int rtbwidth;
          DrawBar(WinLeft,WinTop,WinRight,WinDepth,DRAW);
          gotoxy(5,3);
          cout<<"real time data is selected......";
```

```
            gotoxy(5,5);
            cout<<"Enter the required bandwidth\t";
            cin>>rtbwidth;
            gotoxy(5,7);

            cout<<"Admission control checking for"<<rtbwidth<<"kbps bandwidth......";
            rtadmission_check(rtbwidth);
            getch();
            main();
            break;
case 3:
            int ntbwidth;
            DrawBar(WinLeft,WinTop,WinRight,WinDepth,DRAW);
            gotoxy(5,3);
            cout<<"non real time data is selected......";
            gotoxy(5,5);
            cout<<"Enter the required bandwidth\t";
            cin>>ntbwidth;
            gotoxy(5,7);

            cout<<"Admission control checking for "<<ntbwidth<<" kbps bandwidth......";
            ntadmission_check(ntbwidth);
            getch();
            main();
            break;


default:  main();


            }
}
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void vadmission_check(float voice)
{
 float v;
 v=AVAILABLE_BANDWIDTH*VOICECALL_BANDWIDTH;
 if(voice<v || voice<FCFS*AVAILABLE_BANDWIDTH)
 {
            gotoxy(5,10);
            delay(800);
            cputs("ADMISSION IS ALLOWED...");
            }
else
{
gotoxy(5,10);
delay(800);
cputs("\nADMISSION IS NOT ALLOWED....");
}

}


//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////
void rtadmission_check(int rt)
{
int rtbwidth_allocated=0;
int rtbwidth_needed=rtbwidth_allocated+rt;
int x= ((1-HANDOVER_BANDWIDTH)*(AVAILABLE_BANDWIDTH)-(rtbwidth_allocated));
int y=(1-HANDOVER_BANDWIDTH)*(AVAILABLE_BANDWIDTH-rtbwidth_allocated);
if(rtbwidth_needed<=x && rtbwidth_needed<=y)
 {
        gotoxy(5,10);
        delay(800);
        cputs("ADMISSION IS ALLOWED...");
        }
else
{
gotoxy(5,10);
delay(800);
cputs("\nADMISSION IS NOT ALLOWED....");
}

}

//////////////////////////////////////////////////////////////////////////////////////////////////////////
void ntadmission_check(int nt)
{
int nrtql;
nrtql=THRESHOLD*NRTQ_BUFFER;
if(nt<=nrtql)
        {
        gotoxy(5,10);
        delay(800);
        cputs("ADMISSION IS ALLOWED...");
        }
else
{
gotoxy(5,10);
delay(800);
cputs("\nADMISSION IS NOT ALLOWED....");
}
}

//////////////////////////////////////////////////////////////////////////////////////////////////////////
void TutAdmin(void )
  {
  int i ,SubChoice;
  clrscr();
  DisplayMenu(5,1,TutMenu);
  StructLen=GetStructLen(TutMenu);
  SubChoice= GetChoice(StructLen);
  DrawBar(WinLeft,WinTop,WinRight,WinDepth,DELETE);
  DoSubChoice1(SubChoice);
  }
//////////////////////////////////////////////////////////////////////////////////////////////////////////
void DoSubChoice1(int Choice)
{
 int i;
```

```
gotoxy(8,12);
switch(Choice)
{
 case 1:
          clrscr();
          DrawBar(WinLeft,WinTop,WinRight,WinDepth,DRAW);
          gotoxy(5,5);
          cputs("UMTS is the UniversalMobile Telecommunication System Networks.");
          gotoxy(5,6);
          cputs("It works on the principle of W-CDMA.");
          gotoxy(5,7);
          cputs("It does have the capability to transfer audio,video,");
          gotoxy(5,8);
          cputs("graphical,textual etc. on mobile .");
          getch();
          main();
          break;
 case 2:
          clrscr();
          DrawBar(WinLeft,WinTop,WinRight,WinDepth,DRAW);
          gotoxy(5,5);
          cputs("The UMTS is a combination of core network and UTRAN ");
          gotoxy(5,6);
          cputs("UTRAN consists of a set of radio network controllers");
          gotoxy(5,7);
          cputs("that are connected to the core network.");
          gotoxy(5,8);
          cputs("The core network comprises of the same supporting nodes as in GSM.");
          gotoxy(5,9);
          cputs("the RNC is responsible for control of the connected nodes");
          gotoxy(5,10);
          cputs("i.e Transceiver stations and the radio link to the mobile station.");
          getch();
          main();
         break;
 case 3:
          clrscr();
          DrawBar(WinLeft,WinTop,WinRight,WinDepth,DRAW);
          gotoxy(5,5);
          cputs("QoS for UMTS are specified into four classes.");
          gotoxy(5,6);
          cputs("CONVERSATIONAL CLASS:Real Time Traffic(Most delay sensitive).");
          gotoxy(5,7);
          cputs("STREAMING CLASS:One Way Real Time Traffic.");
          gotoxy(5,8);
          cputs("INTERACTIVE CLASS:Are Mainly Used For Internet Application.");
          gotoxy(5,9);
          cputs("BACKGROUND CLASS:file Dowanloading and have less priority than IC.");
          getch();
          main();
         break;
 case 4:
          clrscr();
          DrawBar(WinLeft,WinTop,WinRight,WinDepth,DRAW);
          gotoxy(5,5);
          cputs("Before a mobile session starts, the mobile user specify its");
```

```
            gotoxy(5,6);
            cputs("traffic characteristics and desired performance requirements");
            gotoxy(5,7);
            cputs("which is called as QoS profile.");
            gotoxy(5,8);
            cputs("The admission controler then accept or reject the session");
            gotoxy(5,9);
            cputs("according to QoS profile and the current network state. ");
            getch();
            main();
            break;
case 5:

        int gd = DETECT, gm;
        initgraph(&gd,&gm,"d:\\tc\\bgi");
        cleardevice();
        setbkcolor(3);
        setcolor(RED);
        rectangle(270,10,400,35);
        outtextxy(285,15,"core network");
        rectangle(45,60,635,285);
        rectangle(75,125,270,85);
        outtextxy(80,100,"Radio Network Controller");
        rectangle(400,125,595,85);
        outtextxy(405,100,"Radio Network Controller");
        outtextxy(325,150,"UTRAN");
        rectangle(50,200,150,250);
        outtextxy(75,225,"Node B");
        rectangle(200,200,300,250);
        outtextxy(225,225,"Node B");
        rectangle(350,200,450,250);
        outtextxy(375,220,"Node B");
        rectangle(530,200,630,250);
        outtextxy(555,225,"Node B");
        rectangle(250,330,450,350);
        outtextxy(300,335,"Mobile Station");
        delay(1500);
        line(335,35,173,85);
        delay(1500);
        line(335,35,498,85);
        delay(1500);
        line(170,125,80,200);
        delay(1500);
        line(170,125,250,200);
        delay(1500);
        line(498,125,400,200);
        delay(1500);
        line(498,125,580,200);
        delay(1500);
        line(330,260,300,315);
        delay(1500);
        line(360,260,330,315);
        delay(1500);
        line(330,260,330,315);
        delay(1500);
        line(270,105,400,105);
        delay(1500);
        outtextxy(250,425,"UMTS ARCHITECTURE");
        getch();
        closegraph();
        main();
        break;

default:
        main();
 }
 }

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void DrawBar(int left,int top,int right,int bottom,int fun)
{
 int lenght,width,line;
 int xax,yax;
 switch(fun)
 {
  case DRAW:
         gettext(1,1,80,25,buffer1);
         window(left,top,right,bottom);
         textbackground(BLUE);
         clrscr();
         textcolor(WHITE);
         lenght=bottom-top+1;
         width=right-left+1;
           for(xax=1;xax<=width;xax++)
             for(yax=1;yax<=lenght;yax++)
             {
                 gotoxy(xax,yax);
                 putch(' ');
         }
             gotoxy(1,1);
             putch('É');
             gotoxy(width,1);
             putch('»');
             gotoxy(1,lenght-1);
             putch('È');
             gotoxy(width,lenght-1);
             putch('¼');
             gotoxy(1,lenght-1);insline();
             for(line=2;line<=width-1;line++)
             {
             gotoxy(line,1);
             putch('Í');
             }
             for(line=2;line<=lenght-1;line++)
             {
             gotoxy(1,line);
             putch('º');
             }
             for(line=2;line<=width-1;line++)
             {
                 gotoxy(line,lenght);
                 putch('Í');
```

```
                    }
                    for(line=2;line<=lenght-1;line++)
                    {
                     gotoxy(width,line);
                     putch('º');
                     }

                     break;
  case DELETE:
                     puttext(1,1,80,25,buffer1);
                     break;

   }

}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void display1(void)
 {
        int gd=DETECT,gm,i,j;
        initgraph(&gd,&gm,"d:\\tc\\bgi");
        setcolor(RED);
        setfillstyle(SOLID_FILL,RED);
        rectangle(10,10,getmaxx()-10,getmaxy()-10);
        rectangle(14,14,getmaxx()-14,getmaxy()-14);
        floodfill(12,12,RED);
        scroll(260,35,525,"Thesis On",0,4);
        scroll(180,70,500,"ADAPTIVE PERFORFORMANCE MANAGEMENT ",0,0);
        scroll(150,100,500,"FOR UNIVERSAL MOBILE TELECOMMUNICATION
SYSTEMS",0,0);
        scroll(260,180,500,"submitted by :",0,5);
        scroll(260,220,500,"Devendra Singh",0,0);
        scroll(260,230,500,"M.Tech.(CST)",0,0);
        scroll(260,240,500,"IIT Roorkee.",0,0);
        scroll(35,300,200,"guided by :",0,5);
        scroll(50,340,200,"Dr.Mohan Lal",0,0);
        scroll(50,355,200,"Asst.Professor",0,0);
        scroll(50,365,200,"New Computational Facility",0,0);
        scroll(50,375,200,"IIT Roorkee.",0,0);
        scroll(400,340,500,"Dr. Manoj Misra ",0,0);
        scroll(400,355,500,"Associate Proffessor. ",0,0);
        scroll(400,365,500,"E&C Deptt. ",0,0);
        scroll(400,375,500,"IIT Roorkee. ",0,0);
        getch();
        closegraph();
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void scroll(int x,int y,int maxx,char *str,char enlarge,char font) {
        int i,j,k=0;
        char temp[80];
        int poly[10];
        settextstyle(font,0,1);
        temp[0]=0;
        for(i=maxx;i>x;i-=D) {
                setcolor(0);
                if(kbhit()||flag) {
                        flag=1;
                        return;
                }
                if(i%5==0) k++;
                if(k!=strlen(temp)&&k<=strlen(str)) {
                        strncpy(temp,str,k);
                        temp[k]=0;
                }
                outtextxy(i+D,y,temp);
                setcolor(GREEN);
                outtextxy(i,y,temp);
                delay(5);
        }
        setfillstyle(SOLID_FILL,0);
        poly[0]=50; poly[1]=70;
        poly[2]=500;poly[3]=70;
        poly[4]=500;poly[5]=100;
        poly[6]=50; poly[7]=100;
        poly[8]=50; poly[9]=70;
        if(enlarge) {
                for(k=2;k<5;k++) {
                        setcolor(0);
                        fillpoly(5,poly);
                        if(kbhit()||flag) {
                                flag=1;
                                return;
                        }
                        settextstyle(font,0,k);
                        setcolor(GREEN);
                        outtextxy(i-(k-2)*35,y,str);
                        delay(150);
                }

        }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
/**************************************************Queue.h**************************************************/
#include<iostream.h>
template<class T>
struct list{
T item;
int id;
list* next;
};
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
template<class T>
class queue{
protected:
        list<T>* front, *rear,*qptr;
public:
        int length,qid;
        queue();
        ~queue();
        int isEmpty();
        void add(T);
        int remove(T&);
        int del(int,int&);
        void go();
        void set(){qptr=rear;}
        void print();
};
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
template<class T>
 queue<T>::queue(){
qptr=NULL;front=NULL;rear=NULL;
length=0;
}

template<class T>
queue<T>::~queue(){
cout<<endl<<"queue destructor";
}
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
template<class T>
void queue<T>::add(T i){
list<T>* newptr=new list<T>;
newptr->item=i;
//newptr->id=length;
++length;
if(isEmpty()){
        front=rear=newptr;
        newptr->next=NULL;
        qid=0;
}
else {
        newptr->next=rear;
        rear=newptr;
        qid++;
        }
newptr->id=qid;
}
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
template<class T>
int queue<T>::isEmpty(){
if( front==NULL)
        return 1;
else    return 0 ;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
template<class T>
int queue<T>::remove(T& val){
list<T>*temp=rear;
if (isEmpty())
        return 0;
else{
        if (front==rear)
                front=rear=NULL;
        else{
        while(temp->next!=front)
                temp=temp->next;
        front=temp;
        temp=temp->next;
        front->next=NULL;
        }
  }
  val=temp->item;
  delete temp;
  --length;
return 1;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
template<class T>
int queue<T>::del(int ind,int &call_type){
list<T>*temp,*ptr;
temp=ptr=rear;
if(ptr == NULL)
cout<<rear;
if (isEmpty())
        return 0;//delete fail
else{
        while( (ptr->id!=ind)&& (ptr!=NULL))
                ptr=ptr->next;
  }
if(ptr==NULL) {cout<<endl<<"Fatal Error:delete fail,id not exist";
                        exit(1);}
           if( (ptr==rear) && (ptr==front) ){// 1) only one node
                ptr=front=rear=NULL;
                }

           else if(ptr==rear){ // 2) ptr points to the rear node
                rear=rear->next;
                ptr->next=NULL;
                ptr=NULL;
                }
           else if(ptr==front){ // 3) ptr points to the last node
                while(temp->next!=front)
                   temp=temp->next;
```

```
                    front=temp;
                    temp=ptr;
                    front->next=NULL;
                    ptr=NULL;
                    }
             else{
                    while(temp->next!=ptr)
                    temp=temp->next;
                    temp->next=ptr->next;
                    temp=ptr;
                    ptr=NULL;
                    temp->next=NULL;
                    }
   call_type=temp->item.type;
   delete temp;
   --length;
return 1;
}
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
template <class T>
void queue<T>::go(){
 if(qptr==NULL)
         qptr=rear;

 if(qptr==front)
         qptr=rear;
 else
         qptr=qptr->next;
}
template <class T >
void queue<T>::print(){
}
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
/*********************************************************Rand.h*********************************************/
#define MODULS 2147483647
#define MULT1 24112
#define MULT2 26143
/* set default seed for 100 streams */
static long   zrng[] ={
193272912, 281629770,20006270,1280689831, 2096730329,1933576050
};

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//gnerate next random
float rand (int stream)
{
 long zi, lowprd, hi31;
 zi    =zrng[stream];
 lowprd=(zi& 65535) * MULT1;
 hi31  =(zi >> 16) * MULT1 + (lowprd >> 16);
 zi    =((lowprd & 65535)- MODULS) + ((hi31  & 32767) << 16) + (hi31 >> 15);
 if (zi<0) zi+=MODULS;
 lowprd=(zi & 65535)*MULT2;
 hi31  =(zi>>16) * MULT2 + (lowprd >> 16);
 zi    =((lowprd & 65535) - MODULS ) + (( hi31 & 32767) << 16) + ( hi31 >> 15);
 if (zi<0)  zi+=MODULS;
 zrng[stream] = zi ;
 return ((zi >> 7 | 1) + 1) / 16777216.0;

  }
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// set the current zrng for stream "stream" to zset
void randst(long zset, int stream)
{
zrng[stream]= zset;
}

//retun the current zrng for stream "stream"
long randgt(int stream)
{
  return zrng[stream];
}


//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
/**********************************************************Sml.h**********************************************/
int resRadio;
#define max_channels 30
#define WRR   1
#define WFQ   2

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
struct calls{
  float atime;
  float end;
  int type;
};
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
struct handover{
  float atime;
  float priorty;
  float q_time;
  int type;
};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
class Random{
protected:
        float uniform(float,float);
        int int_uniform(int);
        float expon(float);
        int random_integer(float prob_dist[]);
};
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
template <class T>
class ho_queue:public queue <class T>{
public:
        T get(){return (qptr->item);}
        void del();
        void m_add(T);

};
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
template <class T>
void ho_queue<T>::m_add(T i){

        {
                float val;
                list<T>* newptr=new list<T>;
                list<T>* temp=rear;
                newptr->item=i;
                val=i.q_time;
                ++length;
                if(isEmpty()){
                        front=rear=newptr;
                        newptr->next=NULL;
                        qid=0;
                }
                else {

                         if(rear->item.q_time<=val){
                                newptr->next=rear;
                                rear=newptr;
                                qid++;
                        }
                        else if(front->item.q_time>val){
                                front->next=newptr;
                                front=newptr;
                                front->next=NULL;
                                qid++;
                        }

                        else{
                                while( (temp->next->item.q_time>val)&&(temp->next-
>next!=NULL) )

                                        temp=temp->next;
                                newptr->next=temp->next;
                                temp->next=newptr;
                                qid++;
                                }
                        }

                newptr->id=qid;
        }
}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
template <class T>
void ho_queue<T>::del(){
        list<T>*temp=rear;

        if( (qptr==rear) && (qptr==front) ){
                qptr=front=rear=NULL;
                }

        else if(qptr==rear){
                rear=rear->next;
                qptr->next=NULL;
                qptr=rear;
                }

        else if(qptr==front){
                while(temp->next!=front)
                   temp=temp->next;
                front=temp;
                temp=temp->next;
                front->next=NULL;
                qptr=rear;
                }

        else{
                while(temp->next!=qptr)
                   temp=temp->next;
                temp->next=qptr->next;
                temp=qptr;
                qptr=qptr->next;
                temp->next=NULL;
                }
            delete temp;
            --length;

    }

////////////////////////////////////////////////////////////////////////////////////////////////////
template<class T>
class call_queue:public queue <class T>{
public:
        void get(int&id,T&call){id=qptr->id ; call=qptr->item;}

};

////////////////////////////////////////////////////////////////////////////////////////////////////
class Events:public Random{
public:
        float clock,next_call,next_handover,ho_delay,miat,hmiat;
        int busy_channels, next_event_type,max_q_len;
        int scheme;
        int call_id;
        long int total_calls,blocked,new_success,q_len,ho_success,
                 ho_fail;
        calls call;
        call_queue<calls> call_list;
        handover ho;
        ho_queue<handover> q_ho;
        Events();
        void new_call();
        void new_handover();
        void release_channel(int);
private:
        void q_scan();



};
////////////////////////////////////////////////////////////////////////////////////////////////////
class simulation: public Events{
public:
        simulation();
        simulation(int,char*);
        ~simulation(){fout.close();}
        void start();
        void start(long int);
        void traffic(float,int);
        void report();
        void save();
private:

        float load;
        ofstream fout;

        void timing();
        void intialize();
```

```
};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Member functions implementation of Random class */
float Random::expon(float mean){
float u;
do{/**/u=rand(2);
}while(u==0);
float exp_u=-mean*log(u);
return exp_u;//-mean*log(u);
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Member functions implementation of Events class */
Events::Events(){
 next_call=0;
 clock=0;
 busy_channels=0;
 blocked=0;
 new_success=0;
 ho_success=0;
 q_len=0;
 ho_delay=0;
 ho_fail=0;
 max_q_len=0;
 total_calls=0;
 call_list.length=0;
 q_ho.length=0;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void Events::new_call(){
 next_call=expon(miat)+clock;
 total_calls++;
 if(busy_channels + resRadio <max_channels)
 {
   busy_channels++;
          call.end=expon(mcl)+clock;
   call.atime=clock;

   if (call_list.length > max_channels) {
                 cout<<"\n\n new call :calls more than channels";
                 getch();
                 exit(1);
                 }

   call_list.add(call);

   new_success++;
   }
 else{
       blocked++;

   }

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
void Events::new_handover(){
  next_handover=expon(hmiat) +clock;
 total_calls++;
 if(busy_channels <max_channels){
    busy_channels++;
    call.end=expon(hmcl)+clock;
    call.atime=clock;
if (call_list.length > max_channels) {
                    cout<<"\n\handover :calls more than channels";
                    getch();
                    exit(1);
                    }
    call_list.add(call);
     ho_success++;
     ho_delay+=0;
        }
else{

         if(   (scheme==WRR)||(scheme==WFQ)  ){
          ho_fail++;
         }

         else{

            ho.atime=clock;
            ho.priorty=0;
            ho.q_time=expon(max_in_q);

         if (q_ho.length > 999) {
                    ho_fail ++;
                    }


            q_len++;
            if( max_q_len < q_ho.length ) max_q_len=q_ho.length;
         }
   }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void Events::release_channel(int ind){
handover ho;
         int ct;
         call_list.del(ind,ct);
         busy_channels--;
         if(!q_ho.isEmpty())
           while(  (!q_ho.isEmpty()) && (busy_channels<max_channels)  )
         .. {

             q_scan();
                  if(!q_ho.isEmpty()){
                  q_ho.remove(ho);
                  busy_channels++;
                  call.end=expon(hmcl)+clock;
                  call.atime=clock;
                  if (call_list.length > max_channels) {
                          cout<<"\n Q handover :calls more than channels";
```

```
                        getch();
                        exit(1);
                    }
                call_list.add(call);
                ho_success++;
                ho_delay+=clock-ho.atime;
               }
        }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void Events::q_scan(){
 handover ho;
 int id, len;
 q_ho.set();
 len=q_ho.length;
 for(int i=0;i<len;i++){
        ho=q_ho.get();
        if ( (clock-ho.atime)> ho.q_time/*max_in_q/*expon(max_in_q)*/){

                ho_fail ++;
                q_ho.del();
            }
        else
            q_ho.go();
        }
}
simulation::simulation(){
 fout.open("ho.txt");
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
simulation::simulation(int s,char* fileName){
 scheme=s;
         fout.open(fileName);
         if(!fout){
                cout<<"can't open '"<<fileName<<"'.";
                exit(0);
                getch();
        }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void simulation::start(){
next_call=0;
next_handover=0;
intialize();
while(total_calls<MAX_SAMPLES ){
        timing();
        switch(next_event_type){
                case NEWCALL :  new_call();break;
                case HANDOVER:  new_handover();break;
                case RELEASE :  release_channel(call_id);break;
                }
        }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void simulation::traffic(float l,int percentage ){
 load=l;
 miat=(float(load)/100) * percentage;
 hmiat=(float(load)/100) * (100-percentage);
 miat=3600/miat;
 hmiat=3600/hmiat;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void simulation::report(){
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void simulation::save(){
long int processed= new_success+blocked+ho_success+ho_fail;
fout<<"load"<<load<<'\t ';
fout<<"blocked"<<(float(blocked)/float(processed) ) <<'\t ';
fout<<"handover failure"<<(float(ho_fail)/float(processed) ) <<'\t ';
fout<<"success"<<(float(ho_success + new_success) /float(processed) );
fout<<endl;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void simulation::intialize(){
  Events::Events();
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void simulation::timing(){
 float min_time_event=1.0e30;
 calls tmpcall;
 int id,len;
 next_event_type=RELEASE;
 call_list.set();
 len=call_list.length;
 for(int i=0;i<len/*call_list.length*/;i++){
          call_list.get(id,tmpcall);
          if (tmpcall.end < min_time_event ){
                  min_time_event=tmpcall.end;
                  call_id=id;

                  }
            call_list.go();
          }
 if(next_call<min_time_event){
          min_time_event=next_call;
          next_event_type=NEWCALL;
          }
 if(next_handover<min_time_event){
          min_time_event=next_handover;
          next_event_type=HANDOVER;
          }
 clock=min_time_event;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////// //////////          UMTS queue          ////////////////
struct Header
{
 int source ;
 int destination ;
 int gen_time ;
 int serviced ;
} ;
struct UMTS_cell
{
 Header header ;
 int data ;
} ;

struct Queue_cell
{
 UMTS_cell atm_cell ;
 Queue_cell *next ;
} ;
struct Queue
{
 Queue_cell *front ;
 Queue_cell *rear ;
 long cells_present ;
 long cells_passed ;
 long cells_lost ;
} ;

void init_queue(Queue *p)
{
 p->front = NULL ;
 p->rear = NULL ;
 p->cells_present = 0 ;
 p->cells_passed = 0 ;
 p->cells_lost = 0 ;
}
int is_empty(Queue *p)
{
 if( (p->front==NULL)&&(p->rear==NULL) )
   return 1 ;
 else
   return 0 ;
}
void enter_queue(Queue *p,UMTS_cell a_cell)
{
 Queue_cell *q_cell = new Queue_cell ;
 q_cell->atm_cell = a_cell ;
 q_cell->next = NULL ;
 if( (p->front)==NULL && (p->rear)==NULL )          // queue is empty
    p->front = q_cell ;
 else
    (p->rear)->next = q_cell ;
 p->rear = q_cell ;
 p->cells_present++ ;
}
UMTS_cell leave_queue(Queue *p)
```

```
{
 Queue_cell *temp ;
 UMTS_cell a_cell ;
 temp = p->front ;
 if( (p->front==NULL) && (p->rear==NULL) )
 {
   cout<<"\nQueue is empty ! " ;
   exit(1) ;
 }
 if( (p->front) == (p->rear) )
  {
   (p->front) = NULL ;
   (p->rear) = NULL ;
  }
 else
   p->front = (p->front)->next ;
 a_cell = temp->atm_cell ;
 p->cells_present-- ;
 delete temp ;
 return a_cell ;
}
```

```
//////////////////// cell loss probability ////////////////////////////////////

#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include"d:\tc\bin\dev\queue.h"

#define IP_LENGTH 1000
#define OP_LENGTH 1000
long tot_cells_lost_at_op();
long tot_cells_lost_at_ip();

Queue *ip_queue[10] ;
Queue *op_queue[10] ;

float tot_cells_generated=0.0;
int temp[10];
long total_loss=0.0;
int dest[10];

UMTS_cell gen_UMTS_cell(int sou,int des)
{
  UMTS_cell a_cell ;

  (a_cell.header).source = sou ;
  (a_cell.header).destination = des ;
   (a_cell.header).serviced = 0 ;
  tot_cells_generated++ ;
  return a_cell ;
}


 for( int i=0 ; i<10 ; i++ )
  {
        dest[i]=random(10) ;
  }
  for(i=0;i<10;i++)
  {
           UMTS_cell a_cell=gen_UMTS_cell(i,dest[i]) ;
              if( (ip_queue[i]->cells_present)<IP_LENGTH )
                         enter_queue(ip_queue[i],a_cell) ;
           else
                     (ip_queue[i]->cells_lost)++ ;

if( (op_queue[i]->cells_present)<OP_LENGTH )
         {
           enter_queue(op_queue[i],(((ip_queue[temp[i]])->front)->atm_cell)) ;
           break ;
          }

         else
          (op_queue[i]->cells_lost)++ ;

}
```

```
total_loss=tot_cells_lost_at_ip()+tot_cells_lost_at_op();
float clp=(tot_cells_generated-total_loss)/tot_cells_generated;
cout<<clp;

}



long tot_cells_lost_at_op()
{
 long tot=0 ;
 for( int i=0;i<10;i++ )
 {
  tot+=op_queue[i]->cells_lost;

 }
 return (tot) ;
}

long tot_cells_lost_at_ip()
{
 long tot=0 ;
 for( int i=0 ; i<10 ; i++ )
 {
  tot+=ip_queue[i]->cells_lost ;
 }
 return (tot) ;
}
```

```
///////////////// simulation file ////////////////////////////////

int resRadio;
int Total_Bandwidth=7680;
//#include"d:\tc\bin\umtspkt.h"
#define max_channels 30
#define MAX_SAMPLES 100000
#define HANDOVER_BANDWIDTH 0.5
#define THRESHOLD 0.9
#define WRR   1
#define WFQ   2


struct calls{
  float atime;
  float end;
  int type;
};
struct handover{
  float atime;
  float priorty;
  float q_time;
  int type;
};


class Random{
protected:
          float uniform(float,float);
          int int_uniform(int);
          float expon(float);
          int random_integer(float prob_dist[]);
};

template <class T>
class ho_queue:public queue <class T>{
public:
          T get(){return (qptr->item);}
          void del();
          void m_add(T);

};
template <class T>
void ho_queue<T>::m_add(T i){

          {
                    float val;
                    list<T>* newptr=new list<T>;
                    list<T>* temp=rear;
                    newptr->item=i;
                    val=i.q_time;
                    ++length;
                    if(isEmpty()){
                              front=rear=newptr;
                              newptr->next=NULL;
                              qid=0;
```

```
            }
            else {

                    if(rear->item.q_time<=val){
                        newptr->next=rear;
                        rear=newptr;
                        qid++;
                    }
                    else if(front->item.q_time>val){
                        front->next=newptr;
                        front=newptr;
                        front->next=NULL;
                        qid++;
                    }

                    else{
                            while( (temp->next->item.q_time>val)&&(temp->next-
>next!=NULL) )

                                temp=temp->next;
                            newptr->next=temp->next;
                            temp->next=newptr;
                            qid++;
                        }
                    }

            newptr->id=qid;
    }
}



template <class T>
void ho_queue<T>::del(){
        list<T>*temp=rear;

        if( (qptr==rear) && (qptr==front) ){
            qptr=front=rear=NULL;
            }

        else if(qptr==rear){
            rear=rear->next;
            qptr->next=NULL;
            qptr=rear;
            }

        else if(qptr==front){
            while(temp->next!=front)
              temp=temp->next;
            front=temp;
            temp=temp->next;
            front->next=NULL;
            qptr=rear;
            }

        else{
            while(temp->next!=qptr)
```

```
                        temp=temp->next;
                     temp->next=qptr->next;
                     temp=qptr;
                     qptr=qptr->next;
                     temp->next=NULL;
                     }
                delete temp;
                --length;

        }



        template<class T>
        class call_queue:public queue <class T>{
        public:
                void get(int&id,T&call){id=qptr->id ; call=qptr->item;}

        };

        class Events:public Random{

        public:
                float clock,next_call,next_handover,ho_delay,miat,hmiat,sum,sumof_bandwidth;
                int busy_channels, next_event_type,max_q_len,num;
                int scheme;
                int call_id;
                int clp;
                long int total_calls,blocked,new_success,q_len,ho_success,
                        ho_fail;
                calls call;
                call_queue<calls> call_list;
                handover ho;
                ho_queue<handover> q_ho;
                Events();
                void new_call();
                void new_handover();
                void release_channel(int);
        private:
                void q_scan();



        };

        class simulation: public Events{
        public:
                simulation();
                simulation(int,char*);
                ~simulation(){fout.close();}
                void start();
                void get_packet_size(int);
                void bandwidth_utilisation(int);
                void p_mib();
                void start(long int);
                void traffic(float,int);
        void report();
        void save();

private:

        float load;
        ofstream fout;
        void timing();
        void intialize();

};
void simulation::p_mib()
{
float NEW_HANDOVER_BANDWIDTH[30],NEW_THRESHOLD[30];
for(int i=0;i<30;i++)
{
 NEW_HANDOVER_BANDWIDTH[i]=ho_fail*blocked*HANDOVER_BANDWIDTH;
 NEW_THRESHOLD[i]=clp*THRESHOLD;
}
int NRT1=400;
int NRT2=200;
int NRT3=100;
 float w=4*sqrt(NRT1)+2*sqrt(NRT2)+sqrt(NRT3);
 float w1=4*sqrt(NRT1)/w;
 float w2=2*sqrt(NRT2)/w;
 float w3=sqrt(NRT3)/w;

for(i=0;i<30;i++)
{
 fout<<NEW_HANDOVER_BANDWIDTH[i];
 fout<<NEW_THRESHOLD[i];
}

ho_fail=NEW_HANDOVER_BANDWIDTH[i]/(blocked*HANDOVER_BANDWIDTH);
clp=NEW_THRESHOLD[i]/THRESHOLD;
fout<<ho_fail;
fout<<clp;
}


void simulation::get_packet_size(int num)
{

float first[100],val[100];
int x;
float l,b;

for(x=1;x<num;x++)
{
first[x]=x*sqrt(2*3.14*5.4139);
b= (log(x)-1.8821);
l= exp((-b*b)/2*5.4139);
val[x]=first[x]*l;

}
```

```
float time[100],sum=0.0;
  for(int i=1;i<num;i++)
  {
  clock_t start[100], end[100];
  start[i] = clock;
  delay(200);
  end[i] = clock;
  time[i]=((end[i] - start[i]) / CLK_TCK);

  sum+=time[i];

  }
//  cout<<"average turnaround time:"<<sum/num;

}

void simulation::bandwidth_utilisation(int numofpkt)
{

float first[100],val[100],sumof_bandwidth=0.0;
int x;
float l,b;

for(x=1;x<numofpkt;x++)
{
first[x]=x*sqrt(2*3.14*5.4139);
b= (log(x)-1.8821);
l= exp((-b*b)/2*5.4139);
val[x]=first[x]*l;
sumof_bandwidth+=val[x];
}
}


/******************************************************************/
/* Member functions implementation of Random class */
float Random::expon(float mean){
float u;
do{
u=rand(2);
}while(u==0);
float exp_u=-mean*log(u);
return exp_u;
}

/* Member functions implementation of Events class */
Events::Events(){
  next_call=0;
  clock=0;
  busy_channels=0;
  blocked=0;
  new_success=0;
  ho_success=0;
  q_len=0;
  ho_delay=0;
  ho_fail=0;
  max_q_len=0;
  total_calls=0;
  call_list.length=0;
  q_ho.length=0;
}
void Events::new_call(){
  next_call=expon(miat)+clock;
  total_calls++;
  if(busy_channels + resRadio <max_channels)
  {
    busy_channels++;
            call.end=expon(miat)+clock;
     call.atime=clock;

     if (call_list.length > max_channels) {
                    cout<<"\n\n new call :calls more than channels";
                    getch();
                    exit(1);
                    }

     call_list.add(call);

     new_success++;
     }
   else{
          blocked++;

     }
 clp=random(11)/10;

 }

 void Events::new_handover(){
   next_handover=expon(hmiat) +clock;
  total_calls++;
  if(busy_channels <max_channels){
     busy_channels++;
     call.end=expon(hmiat)+clock;
     call.atime=clock;
     if (call_list.length > max_channels) {
                    cout<<"\n\handover :calls more than channels";
                    getch();
                    exit(1);
                    }
     call_list.add(call);
     ho_success++;
     ho_delay+=0;
          }
  else{

          {
           ho_fail++;


          //else{
            ho.atime=clock;
            ho.priorty=0;
            ho.q_time=expon(max_in_q);

        if (q_ho.length > 999) {
                ho_fail ++;
                }

            q_len++;
            if( max_q_len < q_ho.length ) max_q_len=q_ho.length;
        }

    }

void Events::release_channel(int ind){
handover ho;
        int ct;
        call_list.del(ind,ct);
        busy_channels--;
        if(!q_ho.isEmpty())
          while(  (!q_ho.isEmpty()) && (busy_channels<max_channels)  )
          {
           q_scan();
                if(!q_ho.isEmpty()){
                q_ho.remove(ho);
                busy_channels++;
                call.end=expon(hmcl)+clock;
                call.atime=clock;

                if (call_list.length > max_channels) {
                        cout<<"\n Q handover :calls more than channels";


                getch();
                        exit(1);
                    }

                call_list.add(call);
                ho_success++;
                ho_delay+=clock-ho.atime;
                }
          }
}

void Events::q_scan(){
  handover ho;
  int id, len;
  q_ho.set();
  len=q_ho.length;
  for(int i=0;i<len;i++){
        ho=q_ho.get();
        if ( (clock-ho.atime)> ho.q_time){

                ho_fail ++;
                q_ho.del();
              }
        else
            q_ho.go();
        }
}
simulation::simulation(){
 fout.open("ho.txt");
}

simulation::simulation(int s,char* fileName){
 scheme=s;

          fout.open(fileName);
          if(!fout){
                  cout<<"can't open '"<<fileName<<"'.";
                  exit(0);
                  getch();
         }
}
void simulation::start(){
next_call=0;
next_handover=0;
intialize();
while(total_calls<MAX_SAMPLES ){
          timing();
          switch(next_event_type){
                  case NEWCALL :  new_call();break;
                  case HANDOVER:  new_handover();break;
                  case RELEASE :  release_channel(call_id);break;
                  }
}
}
void simulation::traffic(float l,int percentage ){
 load=l;



 miat=(float(load)/100) * percentage;
 hmiat=(float(load)/100) * (100-percentage);

 miat=3600/miat;
 hmiat=3600/hmiat;
}

void simulation::report(){
}

void simulation::save(){

long int processed= new_success+blocked+ho_success+ho_fail;

fout<<"load"<<load;
fout<<"CLBP"<<(float(blocked)/float(processed) );
fout<<"HFP"<<(float(ho_fail)/float(processed) );
fout<<"SUCC"<<(float(ho_success + new_success) /float(processed) );
fout<<"agtime:"<<sum/num;
```

```
fout<<"bndutl:"<<(sumof_bandwidth/Total_Bandwidth);
fout<<endl;

}
void simulation::intialize(){
  Events::Events();
}
void simulation::timing(){
 float min_time_event=1.0e30;
 calls tmpcall;
 int id,len;
 next_event_type=RELEASE;
 call_list.set();
 len=call_list.length;
 for(int i=0;i<len;i++){
          call_list.get(id,tmpcall);
          if (tmpcall.end < min_time_event ){
                   min_time_event=tmpcall.end;
                   call_id=id;

                   }
            call_list.go();
          }
 if(next_call<min_time_event){
          min_time_event=next_call;
          next_event_type=NEWCALL;
          }
 if(next_handover<min_time_event){
          min_time_event=next_handover;
          next_event_type=HANDOVER;
          }
 clock=min_time_event;
}
```