Implementation and Evaluation of a QoS-aware Downlink Scheduling Algorithm for LTE Networks

by

Chih-Hao Howard Chang

B.A.Sc. (Hons., Computer Engineering), Simon Fraser University, 2007

Research Project Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Engineering

in the
School of Engineering Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY

Spring 2014

Approval

Name:	**Chih-Hao Howard Chang**
Degree:	**Master of Engineering**
Title:	***Implementation and Evaluation of a QoS-aware Downlink Scheduling Algorithm for LTE Networks***
Examining Committee:	**Chair:** Shahram Payandeh, P.Eng. Professor

Jie Liang, P.Eng.
Senior Supervisor
Associate Professor

Shawn Stapleton
Supervisor
Professor

Date Defended/Approved: April 30, 2014

Abstract

Long Term Evolution (LTE) is becoming the mainstream of the fourth generation standard for high-speed wireless communications for mobile devices. Its radio access for downlink involves allocation of Physical Resource Blocks (PRB). In order to achieve optimal download performance for different applications to satisfy different QoS requirements, the downlink scheduling algorithm in use plays an important role in determining which PRBs and how are they allocated to each flow of bits. Several researches have exploited different scheduling strategies for flows; however, both the frequency and time domain allocations for PRBs should be taken into account. In this project, we implement and evaluate a QoS-aware downlink packet scheduling algorithm for LTE networks known as the Packet Prediction Mechanism (PPM) using the LTE Simulator (LTE-Sim). The PPM consists of three phases. It first utilizes the PRBs effectively in the frequency domain. It then manages queues and predicts the behaviour of future incoming packets based on the current ones in the queue by the concept of virtual queuing. Finally, it incorporates a cut-in process to rearrange the transmission order and discard overdue packets based on the predicted information from the previous phase. The simulation results demonstrate the effectiveness of the PPM scheme in achieving better downlink transmission performance in terms of Throughput, Delay, Fairness Index, Packet Loss Ratio (PLR), and Spectral Efficiency than other downlink schedulers such as Priority First (PF), Modified Largest Weighted Delay First (MLWDF), and Exponential Proportional Fair (EXPPF).

Keywords: Downlink, LTE, LTE-Sim, PPM, QoS, Scheduling

Acknowledgements

I would like to express sincere gratitude towards my senior supervisor, Dr. Jie Liang for inspiring me to study LTE and taking time and assisting me on this project. Many thanks to him for his generous guidance throughout the project. I would also like to thank the Chair, Dr. Shahram Payandeh, and my supervisor, Dr. Shawn Stapleton for serving on my committee and for their valuable advices. Finally, I would like to thank my mother, Merry Lee, for her persistent support and encouragement.

Table of Contents

Approval ii
Partial Copyright Licence iii
Abstract iv
Acknowledgements v
Table of Contents vi
List of Tables viii
List of Figures ix
List of Acronyms x

Chapter 1. Introduction 1
1.1. Overview of LTE 1
1.1.1. SAE Architecture 1
1.2. Downlink Resource Allocation 3
1.2.1. EPS and Radio Bearers 3
1.2.2. Resource Management 4
1.3. Downlink Scheduling 5

Chapter 2. Use of LTE-Sim for Implementation 7
2.1. LTE-Sim’s Packet Flow 8
2.2. LTE-Sim’s Default Downlink Schedulers 9
2.2.1. Priority First Downlink Scheduler 10
2.2.2. Modified Largest Weighted Delay First Downlink Scheduler 11
2.2.3. Exponential Proportional Fair Downlink Scheduler 11
2.3. PPM Downlink Scheduler in LTE-Sim 13
2.3.1. PPM Parameters 13
2.3.2. PPM Class and Methods 16
2.3.3. PPM Phase I – Initial Scheduling for PRBs 18
2.3.4. PPM Phase II – Queue Management and Packet Delay Prediction 18
2.3.5. PPM Phase III – Cut-in Process 24

Chapter 3. Simulation 25
3.1. Simulation Parameters 26
3.2. Simulation Results 27
3.2.1. Throughput 28
3.2.2. Delay 30
3.2.3. Fairness Index 32
3.2.4. Packet Loss Ratio 33
3.2.5. Spectral Efficiency 34

Chapter 4. Conclusion **35**

References **36**
Appendix A. LTE-Sim Source Code 37
PPM Downlink Scheduling Algorithm Implemented 37

List of Tables

Table 1: Channel Quality Indication Index versus Modulation.. 5
Table 2: Parameters and Their Accessor Methods in LTE-Sim for the PPM.................. 16
Table 3: LTE-Sim Simulation Parameters.. 26

List of Figures

Figure 1: A Typical SAE Architecture of LTE 2
Figure 2: Downlink Physical Resource Block 4
Figure 3: Relationship between Flows, Radio Bearer, QoS, MAC Queue in LTE-Sim 8
Figure 4: Three Phases of the PPM Downlink Scheduling Algorithm [4] 13
Figure 5: Scenario A – two situations for continuous out-of-date packets in the queue [4] ... 19
Figure 6: Scenario B – two situations for continuous out-of-date packets in the queue [4] ... 20
Figure 7: LTE-Sim Single Cell with Interference Scenario 25
Figure 8: Throughput Comparison between PF, MLWDF, EXPPF, and PPM 29
Figure 9: Average Cell Goodput (UE speed = 30 km/h) from the PPM paper [4] 29
Figure 10: Delay Comparison between PF, MLWDF, EXPPF, and PPM 30
Figure 11: Average delay time (UE speed = 30 km/h) [4] 31
Figure 12: Fairness Index Comparison between PF, MLWDF, EXPPF, and PPM 32
Figure 13: Packet Loss Ratio Comparison between PF, MLWDF, EXPPF, and PPM 33
Figure 14: Spectral Efficiency Comparison between PF, MLWDF, EXPPF, and PPM 34

List of Acronyms

CBR	Constant Bit Rate
CQI	Channel Quality Indication
EDGE	Enhanced Data rates for GSM Evolution
eNB	evolved Node B
EPS	Evolved Packet System
E-UTRAN	Evolved UMTS Terrestrial Radio Access
EXPPF	Exponential PF
FDD	Frequency Division Duplex
FIFO	First In, First Out
GSM	Global System for Mobile Communications
HSPA	High Speed Packet Access
HSS	Home Subscriber Server
IMS	IP Multimedia Subsystem
IP	Internet Protocol
L2	Layer-2
LTE	Long Term Evolution
LTE-Sim	LTE Simulator
MAC	Media Access Control
MCS	Modulation Code Scheme
MLWDF	Modified Largest Weighted Delay First
MME	Mobility Management Entity
OFDM	Orthogonal Frequency Division Multiplexing
PDN GW	Packet Data Network Gateway
PF	Proportional Fair
PLR	Packet Loss Ratio
PPM	Packet Prediction Mechanism
PRB	Physical Resource Block
QAM	Quadrature Amplitude Modulation
RE	Resource Element
SAE	System Architecture Evolution
S-GW	Serving Gateway

TDD	Time Division Duplex
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
VoIP	Voice over IP

Chapter 1.

Introduction

1.1. Overview of LTE

Long Term Evolution (LTE) is the latest mainstream technology of the fourth generation (4G) standard for high-speed wireless communications for mobile devices and data terminals. It is an evolution of the existing 2G mobile network technologies such as Global System for Mobile Communications (GSM) and Enhanced Data rates for GSM Evolution (EDGE), and 3G cellular systems such as Universal Mobile Telecommunications System (UMTS) and High Speed Packet Access (HSPA). LTE increases the capacity and speed of wireless data networks using a different radio interface based on newly developed digital signal processing techniques and modulations along with improved and simplified core network infrastructure. The infrastructure is simpler because the number of the inter-connected nodes involved in LTE is less than that of the 2G and 3G architectures.

1.1.1. SAE Architecture

The diagram in Figure 1 illustrates a typical System Architecture Evolution (SAE) architecture of the LTE wireless communication standard. The main component in the SAE architecture is the Evolved Packet Core (EPC), which comprises the Serving Gateway (S-GW), Packet Data Network Gateway (PDN GW), and Mobility Management Entity (MME) sub-components to be described in later paragraphs.

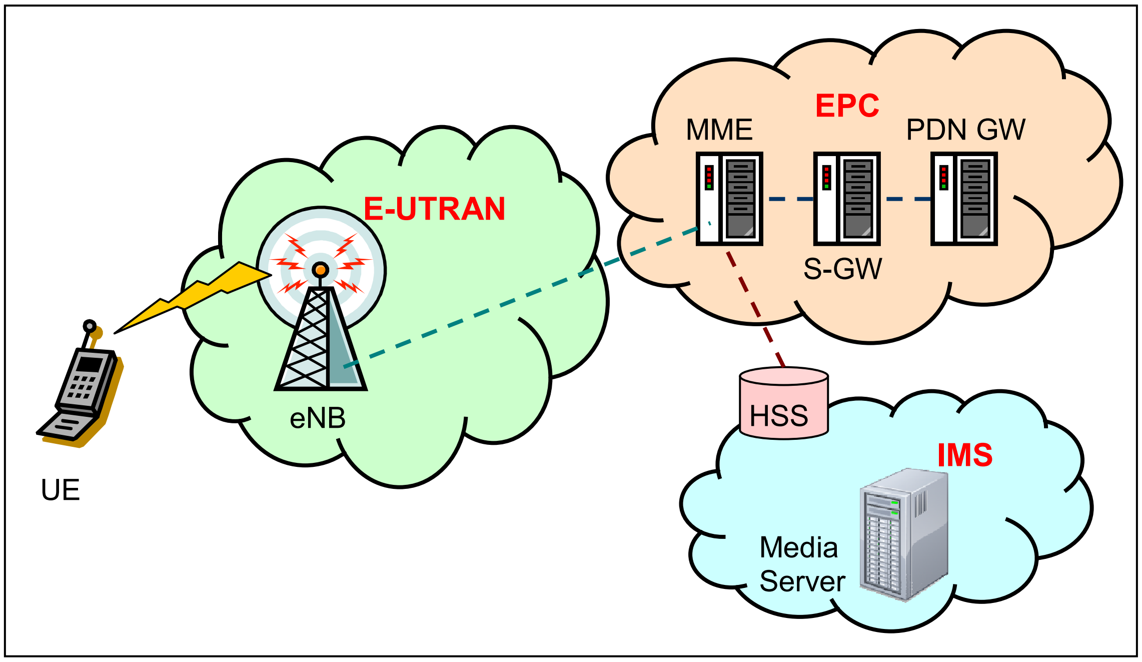

Figure 1: A Typical SAE Architecture of LTE

Two key user-side nodes are the base station, known as the evolved Node B (eNB), and the S-GW. eNBs are connected to the core network over a new air interface called Evolved UMTS Terrestrial Radio Access (E-UTRAN). The S-GW, acting as a local mobility anchor when terminals move across eNBs, is the termination point of the packet data interface towards E-UTRAN [2]. Packets are routed through the S-GW for intra E-UTRAN mobility and mobility with the 2G and 3G technologies.

Next, the User Equipment (UE) is an end-user device such as a mobile phone and a laptop computer with a LTE network adapter, which directly accesses the LTE network via the eNB. The PDN GW that performs policy enforcement, packet filtering, charging support, lawful interception and packet screening, provides connectivity from the UE to external packet data networks by being the UE's point of exit and entry of traffic [2].

The MME is in charge of all the control plane and the capacity signalling functions related to subscriber and session management, including security procedures, terminal-to-network session handling, and idle terminal location management [2]. The

MME authenticates users by interacting with the Home Subscriber Server (HSS) that connects to the packet core network over the IP Multimedia Subsystem (IMS).

The HSS is a central database containing user-related and subscription-related information, with functionalities like mobility management, call and session establishment support, user authentication and access authorization. The IMS is an architectural framework for delivering multimedia services over Internet Protocol (IP) interfaces. A class-based Quality of Services (QoS) concept is incorporated by LTE, allowing service providers to effectively deploy different packet media services [1].

1.2. Downlink Resource Allocation

1.2.1. EPS and Radio Bearers

In LTE, QoS differentiation is provided by an Evolved Packet System (EPS) bearer, which is a virtual connection in the connection-oriented transmission network from the UE to the PDN GW. Before any traffic can be sent between the two endpoints, the EPS bearer must establish a logical channel to provide a data link layer or layer-2 (L2) transport service with specific QoS attributes such as traffic class, bit rate, delivery order, reliability, delay characteristics, priority, etc [3]. The EPS bearer then maps a flow, also known as a bit-stream, into this logical channel. Radio access of the LTE service uses Orthogonal Frequency Division Multiplexing (OFDM) for the downlink, which is the path between the eNB and the UE. LTE supports both Frequency Division Duplex (FDD) and Time Division Duplex (TDD) multiple access techniques. The downlink portion of the logical channel is the radio bearer associated with the EPS bearer. A radio bearer describes the L2 processes that "include such things as prioritization, sequencing, error correction etc. and impact the QoS that is provided for that bit-stream" [3]. The use of the radio bearer allows differentiation of flows with dissimilar QoS requirements to be handled by assigning separate radio bearers to flows with different QoS characteristics [3]. For example, if there are two flows, flow A needs low latency but can tolerate bigger packet losses, whereas flow B does not attach importance to low latency but has a much smaller packet tolerance, they must be achieved by two radio bearers, each of which is configured with the relevant latency and packet loss requirements [3].

1.2.2. Resource Management

The eNB distributes radio resources among downlink flows. Flows are allocated into Physical Resource Blocks (PRBs), which consist of 7 OFDM symbols and 12 sub-carriers in the time-frequency grid as shown in Figure 2. In the frequency domain, the spacing between the sub-carriers is 15 kHz; thus, one PRB spans a total of 180 kHz. In the time domain, one slot is 0.5 ms. There are 84 resource elements (REs) in one PRB. The more PRBs a flow is allocated, the higher the modulation bits will be used in the REs, the higher the code rate.

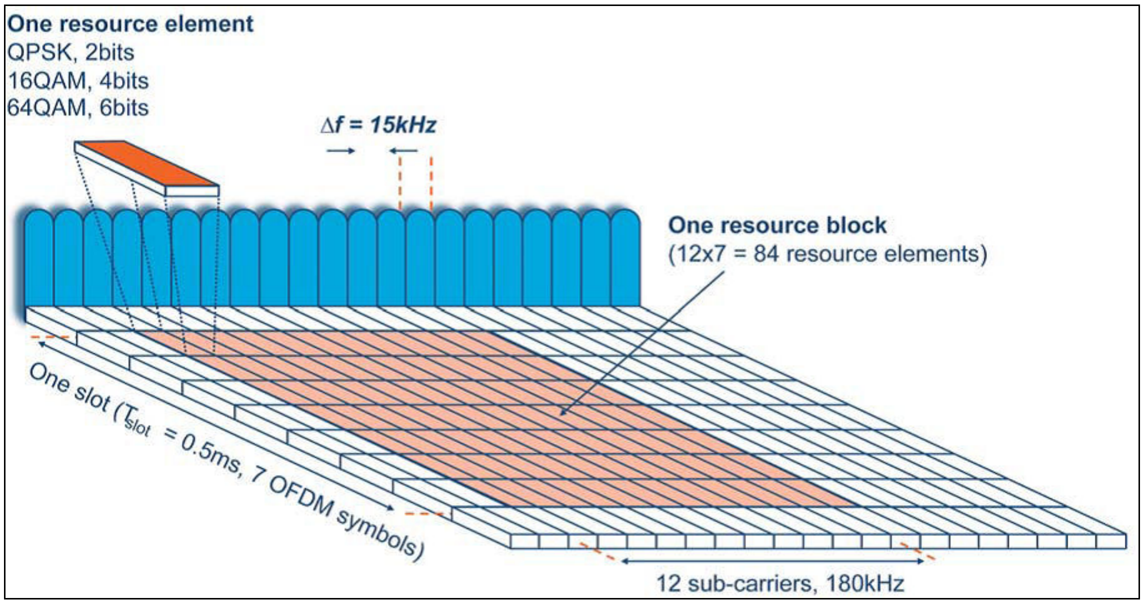

Figure 2: Downlink Physical Resource Block

During the resource allocation process in LTE, three different modulation schemes are provided for a RE to use at run-time by considering the Channel Quality Indication (CQI) feedback that a UE currently experiences and reports to the eNB. One RE can carry either the Quadrature Phase Shift Keying (QPSK), the 16-bit Quadrature Amplitude Modulation (16QAM), or the 64-bit QAM (64QAM) modulated bits. Based on the channel condition of each scheduled flow, the eNB selects the most suitable Modulation Code Scheme (MCS). The CQI reporting mechanism is important since it aims to reduce the packet losses due to channel errors and to improve the transmission efficiency. Table 1 summaries the relationship between the CQI index, which is a 4-bit value ranged from 0000_2 (0_{10}) to 1111_2 (15_{10}), and the corresponding modulation and maximum number of bits that a RE can transmit in the LTE standard.

CQI Index	Modulation	Code Rate (x 1024)	Maximum Number of Bits	Efficiency
0	N/A	N/A	N/A	N/A
1	QPSK	78	2	0.1523
2	QPSK	120	2	0.2344
3	QPSK	193	2	0.3770
4	QPSK	308	2	0.6016
5	QPSK	449	2	0.8770
6	QPSK	602	2	1.1758
7	16QAM	378	4	1.4766
8	16QAM	490	4	1.9141
9	16QAM	616	4	2.4063
10	64QAM	466	6	2.7305
11	64QAM	567	6	3.3223
12	64QAM	666	6	3.9023
13	64QAM	772	6	4.5234
14	64QAM	873	6	5.1152
15	64QAM	948	6	5.5547

Table 1: Channel Quality Indication Index versus Modulation

1.3. Downlink Scheduling

LTE supports scalable channel bandwidths 1.4, 3, 5, 10, 15, and 20 MHz, which represent 6, 15, 25, 50, 75, and 100 PRBs available for allocation, respectively. As can be deduced from Table 1, each of the 84 REs can transmit at most 6 bits within a 0.5 ms symbol time; therefore, each PRB has a transmission bandwidth of 6 x 84 / 0.5 = 1008 kbps. Which PRBs and how are they allocated to each flow at a given point of time depends on the downlink scheduling algorithm in use. The objective is to achieve optimal download transmission performance for different applications, such as video and voice, in order to satisfy different QoS requirements. Reaching an optimal trade-off between utilization and fairness simultaneously is very challenging, especially in the presence of real-time multimedia applications that are characterized by stringent constraints on packet latency and jitter [5]. Therefore, downlink scheduling for resource allocation has been a popular research area. Many scheduling schemes have been studied in recent years, as more and more cellular operators worldwide have started to commercially launch their LTE networks.

In this project, we concentrate on the Packet Prediction Mechanism (PPM) proposed by Wei-Kuang Lai and Chang-Lung Tang in the paper, “QoS-aware Downlink Packet Scheduling for LTE Networks” [4]. It states that, “although many scheduling schemes for flows have been proposed before, simply applying those schemes directly for LTE networks may not achieve good performance.” The PPM is suitable for real-time application services; it formulates both the frequency-domain and time-domain allocations for PRBs into three phases. Phase I is in the frequency domain, which utilizes the PRBs effectively by considering the CQI feedback. Phase II is in the time domain, where PPM first manages queues for different applications and then predicts the packet delays. Finally, phase III involves the use of a cut-in process that rearranges the transmission order and discards those packets that are unable to meet their delay requirements based on the calculated results from phase II. Details of the PPM will be described in later sections.

Chapter 2. Use of LTE-Sim for Implementation

To evaluate the effectiveness of the PPM proposed by [4], we implement this downlink packet scheduling algorithm in LTE-Sim, which is an open-source framework for simulating LTE networks developed by Giuseppe Piro, Luigi Alfredo Grieco, Gennaro Boggia, Francesco Capozzi, and Pietro Camarda. LTE-Sim supports several aspects of LTE networks from the application layer down to the physical layer, such as single-cell and multi-cell environments, QoS management, multi-users environment, user mobility, CQI feedback, handover procedures, and frequency reuse techniques [5]. Fundamental network nodes like UE, eNB, MME and S-GW are modeled in LTE-Sim, with the supports of the trace-based, Voice over IP (VoIP), Constant Bit Rate (CBR), and infinite-buffer traffic generators at the application layer and the management of data radio bearers [5]. Moreover, LTE-Sim comes with the following downlink packet schedulers by default:

- Proportional Fair (PF)
- Modified Largest Weighted Delay First (MLWDF)
- Exponential Proportional Fair (EXPPF)

For the purpose of this project, we compare the simulation results of sending a real-time video stream from the eNB to several UEs using the newly added PPM algorithm with the ones produced by the built-in PF, MLWDF, and EXPPF packet schedulers in LTE-Sim. The trace-based traffic generator that sends video application packets based on a video trace file is used for the simulations with the four downlink scheduling schemes. In addition, the implementation of the PPM requires the understanding of the Radio Bearer, QoS, Media Access Control (MAC) Queue, and Packet components in LTE-Sim.

2.1. LTE-Sim's Packet Flow

In LTE-Sim, the **RadioBearer** class models the radio bearer and activates one when a downlink flow from the **FlowsToSchedule** list starts from the eNB to the UE. For each **RadioBearer** instance, the **QoSParameters** class defines the QoS requirements for the flow. The trace-based traffic generator at the application layer transmits packets (modeled by the **Packet** class) based on a realistic video trace file, which will be transported by a radio bearer. When a video packet is delivered from the trace-based application on the eNB, the packet first goes through the user-plane protocol stack in order to add the User Datagram Protocol (UDP) and (IP) headers, and then gets associated to a particular radio bearer and enqueued at the MAC layer. The resultant IP datagrams are mapped to radio bearers by an IP-based packet classifier. Each radio bearer maintains a First In, First Out (FIFO) queue, modeled by the **MacQueue** class. Figure 3 depicts the relationship of these entities. Eventually, the packet can be sent to the network through the physical layer of the eNB on the logical channel. The UE can then receive the packet from the channel and deliver it to the application layer through the user-plane protocol stack on the UE.

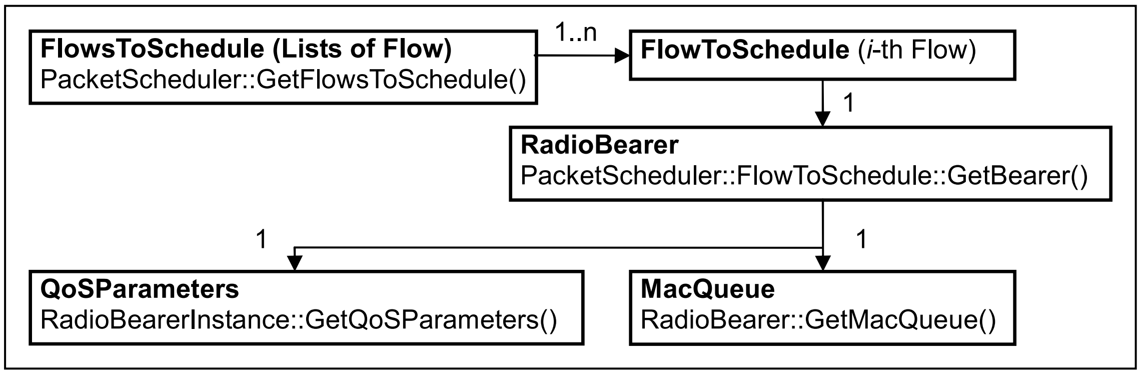
FlowsToSchedule (Lists of Flow)
PacketScheduler::GetFlowsToSchedule()
1..n
FlowToSchedule (i-th Flow)
1
RadioBearer
PacketScheduler::FlowToSchedule::GetBearer()
1
1
QoSParameters
RadioBearerInstance::GetQoSParameters()
MacQueue
RadioBearer::GetMacQueue()

Figure 3: Relationship between Flows, Radio Bearer, QoS, MAC Queue in LTE-Sim

2.2. LTE-Sim's Default Downlink Schedulers

The way that LTE-Sim's downlink schedulers work is that they select all flows that can be scheduled first, and then assign each PRB to the flow with the highest metric. "A flow can be scheduled if an only if it has data packets to transmit at the MAC layer and the receiver UE is not in the idle state. Every TTI, the scheduler computes a given metric for each flow which can be scheduled" [5]. A Transmission Time Interval (TTI) is the duration required for the transmission of data block from higher layers into frames on the radio link. For the sake of combating errors due to fading and interference, data is divided at the transmitter into blocks and the bits within a block are encoded.

Let $m_{i,j}$ or the 2-demonsional array, **metrics[j][i]**, in the LTE-Sim code, be the metric assigned to the *i*-th flow for the *j*-th PRB, the scheduling procedures are as follows [5]:

1. The eNB creates a list of flows that have packets to transmit (**FlowsToSchedule)**. The MAC queue length and CQI feedbacks are stored for each flow in this list.
2. According to the downlink scheduler in use, the chosen metric is computed for each flow in the **FlowsToSchedule** list by calling the function, **<Class name of the specific downlink scheduler in use>::ComputeSchedulingMetric()**.
3. The eNB assigns each PRB to the flow that has the highest metric. As soon as a flow sends all the enqueued packets, it is deleted from the **FlowsToSchedule** list.
4. For each scheduled flow, the eNB computes the size of the quota of data that will be transmitted at the MAC layer during the current TTI. In the end, the eNB invokes dequeueing of packets at the MAC layer for all scheduled flows.

The following sub-sections describe how the built-in LTE-Sim downlink schedulers PF, MLWDF, and EXPPF compute the metric. To calculate for the metric, these schedulers all depend on the average transmission data rate of the *i*-th flow, $\overline{R_i}$, or the value returned by the **RadioBearer::GetAverageTransmissionRate()** function call. This value is updated every TTI based on the following weighted moving average formula:

$$\overline{R_i}(k) = 0.8\overline{R_i}(k-1) + 0.2R_i(k),$$

where $\overline{R_i}(k)$ is the data rate achieved by the *i*-th flow during the *k*-th TTI and $\overline{R_i}(k-1)$ is the data rate in the previous TTI. Next, the schedulers need to know the instantaneous available data rate of the receiver UE for the *j*-th PRB, $r_{i,j}$, or the value returned by **PacketScheduler::FlowToSchedule::GetSpectralEfficiency()** multiplied by 180 kHz, which is the bandwidth of a PRB in the frequency domain. The spectral efficiency here is the information rate in bits/seconds that can be transmitted over the given bandwidth; that is, 180000 Hz in the LTE communication system. It measures how efficient a limited frequency spectrum is utilized, by considering the CQI feedback that the *i*-th flow hosted by the UE has sent for the *j*-th PRB.

2.2.1. Priority First Downlink Scheduler

The PF scheduler is defined in the **DL_PF_PacketScheduler** class in LTE-Sim. Its goal is "to maximize the total network throughput and to guarantee fairness among flows," [5] making it a good choice for non-real-time traffic [4]. The scheduler assigns radio resources taking into account of both the experienced channel quality and the past user throughput [6]. The metric is defined as:

$$m_{i,j} = \frac{r_{i,j}}{\overline{R_i}},$$

or,

$$\frac{spectralEfficiency(i,j) \times 180000}{averageTransmissionRate(i)},$$

in the function **DL_PF_PacketScheduler::ComputeSchedulingMetric()**.

2.2.2. Modified Largest Weighted Delay First Downlink Scheduler

The MLWDF scheduler is defined in the **DL_MLWDF_PacketScheduler** class in LTE-Sim. It is the channel-aware extension of the LWDF policy, which aims at avoiding deadline expiration for real-time operating system and wired networks [7]. MLWDF provides bounded packet delivering delay and prioritizes real-time flows with the highest delay for their head of line (HOL) packets (the first packet to be transmitted in the queue) and the best channel condition by the following metric [5]:

$$m_{i,j} = -\frac{\log \delta_i}{\tau_i} D_{HOL,i} \frac{r_{i,j}}{\overline{R_i}},$$

or,

$$-\frac{\log(drop\Pr obability(i))}{\max Delay(i)} HOLPacketDelay(i) \frac{spectralEfficiency(i,j) \times 180000}{averageTransmissionRate(i)}$$

in the function **DL_MLWDF_PacketScheduler::ComputeSchedulingMetric()**. δ_i is the maximum probability that the HOL packet delay, $D_{HOL,i}$, exceeds the delay threshold, τ_i, for the *i*-th real-time flow. In the LTE-Sim code, the values of these three parameters can be obtained by **QoSParameters::GetDropProbability()**, **RadioBearer::GetHeadOfLinePacketDelay()**, and **QoSParameters::GetMaxDelay()**, respectively. To avoid bandwidth wasting, packets belonging to a real-time flow are discarded from the MAC queue if they are not transmitted before the expiration of their deadline [5]. For non-real-time flows, the metric reduces to that of the PF scheduler.

2.2.3. Exponential Proportional Fair Downlink Scheduler

The EXPPF scheduler is defined in the **DL_EXP_PacketScheduler** class in LTE-Sim. It is designed to favour real-time traffic flows over non-real-time ones. For real-time flows, the considered metric is computed by:

$$m_{i,j} = \exp\left(\frac{-\frac{\log \delta_i}{\tau_i} D_{HOL,i} - \chi}{1+\sqrt{\chi}} \right) \frac{r_{i,j}}{\overline{R_i}},$$

where δ_i is the maximum probability that the HOL packet delay, $D_{HOL,i}$, exceeds the delay threshold, τ_i, for the *i*-th real-time flow, and:

$$\chi = \frac{1}{N_{rt}} \sum_{i=1}^{N_{rt}} -\frac{\log \delta_i}{\tau_i} D_{HOL,i}$$

with N_{rt} denoting the number of active downlink real-time flows. Similar to the MLWDF case, the metric is reduced to that of the PF scheduler for non-real-time flows, and real-time packets are erased from the MAC queue if they are overdue. In LTE-Sim, the metric is jointly computed in the functions **DL_EXP_PacketScheduler::ComputeAW()** and **DL_EXP_PacketScheduler::ComputeSchedulingMetric()** as follows:

$$AW(i) = -\frac{\log(drop\,\Pr obability(i))}{\max Delay(i)} HOLPacketDelay(i) \frac{spectralEfficiency(i,j) \times 180000}{averageTransmissionRate(i)},$$

$$averageAW = \frac{1}{nbFlows} \sum_{i=1}^{nbFlows} AW(i),$$

$$m_{i,j} = \exp\left(\frac{AW(i) - averageAW}{1 + sqrt(averageAW)} \right) \frac{spectralEfficiency(i) \times 180000}{averageTransmissionRate(i)},$$

where the values of $drop\,\Pr obability(i)$, $HOLPacketDelay(i)$, and $\max Delay(i)$ for the *i*-th flow can be obtained by calling the functions, **QoSParameters::GetDropProbability()**, **RadioBearer::GetHeadOfLinePacketDelay()**, and **QoSParameters::GetMaxDelay()**, respectively.

2.3. PPM Downlink Scheduler in LTE-Sim

The PPM downlink scheduling algorithm proposed in [4] focuses on real-time applications and encompasses the following three phases as illustrated in Figure 4:

I. Initial scheduling for PRBs
II. Queue management and prediction of delays for packets
III. Cut-in process

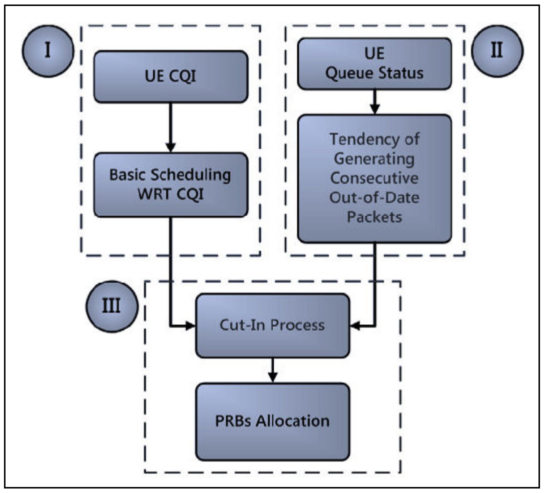

Figure 4: Three Phases of the PPM Downlink Scheduling Algorithm [4]

2.3.1. PPM Parameters

Please note that the term "user" is used throughout the PPM paper; this is really the "flow" in LTE-Sim. Therefore, for the list of parameters to be introduced below, we treat each instance of "user" as "flow" when implementing the PPM algorithm in LTE-Sim. For the convenience of cross-referencing with the variable names that we will use in the LTE-Sim code, the actual variable names are provided in the following table. Their descriptions are directly taken from [4].

Parameter	Variable Name to be used in LTE-Sim	Description
$CQI_{x,y}$	CQI_usr_x_PRB_y	CQI value if PRB *y* is allocated to user *x*
$M_{x,y}$	M_usr_x_PRB_y	Maximum CQI_usr_x_PRB_y for PRB *y*

EU_x	E_Q_k_usr_x	Average throughput for packets of user *x* in queue *k*, which can be obtained by calling **RadioBearer::GetAverageTransmissionRate()**
T_I	T_I	1 TTI = **10 ms**
$DB_{k,i}$	Db_Q_k_pkt_i	Delay bound for packet *i* in queue *k*, which can be obtained by calling **QoSParameters::GetMaxDelay()**
n_k	num_pkts_Q_k	Number of packets in queue *k*, which can be obtained by calling **MacQueue::GetNbDataPackets()**
$t_{k,i}$	ta_Q_k_pkt_i	Arrival time for the *i*-th packet in queue *k*; LTE-Sim does not support it, we add the new method, **Packet::SetTimeArrival()**, for setting the value to the time that the packet is enqueued in **MacQueue::Enqueue()**, and we also add the new method, **Packet::GetTimeArrival()**, for getting the packet arrival time.
$t_{sk,i}$	ts_Q_k_pkt_i	Stamp time for the *i*-th packet in queue *k*, which can be obtained by calling **Packet::GetTimeStamp()**
t_{pi}	tp_pkt_i	Propagation delay in physical layer for packet *i*, which can obtained by the packet size in bits divided by physical channel bandwidth
t_{ok}	to_Q_k	Observation time to see which packets in queue *k* may not arrive at destinations before delay budgets; LTE-Sim does not support it, the new method, **GetTimeObservationForMayNotArrivePackets()** is added in the **MacQueue** class to the retrieve the time when **MacQueue::CheckForDropPackets()** is called to delete the packets from the queue

$R_{k,i}$	R_Q_k_pkt_i	Interval that the *i*-th packet in queue *k* will be scheduled to transmit at; that is, $R_{k,i} = \left\lceil \frac{i+1}{EU_x \times T_I} \right\rceil$
l_k	l_Q_k	First identified packet number in queue *k* whose scheduled transmission time may be late and in consequence the packet does not arrive at the destination in time
h_k	h_Q_k	Last identified packet number in queue *k* (if it exists) whose scheduled transmission time may be late and in consequence the packet does not arrive at the destination in time
L_k	L_Q_k	Estimated loss rate for queue *k*
N_k	N_Q_k	Number of continuously successful transmission packets in queue *k* before the current observation time and after the last dropped packet; LTE-Sim does not support it, we add the new method, **GetNbOfTxPacketsBeforeTimeObservation()**, in the **MacQueue** class for getting the number of the transmitted packets when **MacQueue::CheckForDropPackets()** is called to delete the packets from the queue
T_k	T_Q_k	Threshold for the lost rate of packets in queue *k*, set to **0.5 s** for video applications
ΔT_k	delta_Ta_Q_k	Average inter-arrival time for packets in queue *k*
ΔT_{sk}	delta_Ts_Q_k	Average inter-stamp time for packets in queue *k*
el_k	el_VQ_k	First estimated packet number in the virtual queue of queue *k* whose scheduled transmission time may be late and in consequence the packet does not arrive at the destination in time
eh_k	eh_VQ_k	Last estimated packet number in the virtual queue

		of queue *k* (if it exists) whose scheduled transmission time may be late and cause the packet not to arrive at the destination in time
$ER_{k,n}$	ER_VQ_k_pkt_n	Estimated interval that the *n*-th packet in the virtual queue of queue *k* will be scheduled to transmit at
$et_{k,n}$	eta_VQ_k_pkt_n	Estimated arrival time for the *n*-th packet in the virtual queue of queue *k*
$et_{sk,n}$	ets_VQ_k_pkt_n	Estimated stamp time for the *n*-th packet in the virtual queue of queue *k*
eq_{n_n}	eq_n_VQ_k	Estimated number of new arrival packets in to the virtual queue of queue *k* during the time interval that all n_k packets currently in queue *k* are transmitted
T_{n_k}	T_num_pkts_Q_k	Time needed to transmit all the n_k packets in queue *k*
eL_k	eL_Q_k	Estimated loss rate for incoming packets in queue *k* and its virtual queue
$ET_{x,y}$	ET_usr_x_PRB_y	Expected throughput if PRB *y* is located to user *x*

Table 2: Parameters and Their Accessor Methods in LTE-Sim for the PPM

2.3.2. PPM Class and Methods

For implementing the PPM algorithm in LTE-Sim, a new downlink packet scheduler class called **DL_PPM_PacketScheduler** is created in the new source and header files called:

- /lte-sim/src/protocolStack/mac/packet-scheduler/dl-ppm-packet-scheduler.cpp
- /lte-sim/src/protocolStack/mac/packet-scheduler/dl-ppm-packet-scheduler.h

In addition to the following constructors and methods that inherit from their parent class called **DownlinkPacketScheduler**:

- DL_PPM_PacketScheduler::~DL_PPM_PacketScheduler()

- DL_PPM_PacketScheduler::DL_PPM_PacketScheduler()
- DL_PPM_PacketScheduler::DoSchedule()

the following new methods are defined for the implementation of PPM's three phases:

- **Phase I: Initial Scheduling for PRBs**
  - DL_PPM_PacketScheduler::AllocUserXWithMaxCqiToPrbY(int PRB_y)
    - Allocates user *x* having the best CQI among others to PRB *y*.
- **Phase II: Queue Management and Packet Delay Prediction**
  - DL_PPM_PacketScheduler::ManageQueuesAndPredictPktDelays()
  - DL_PPM_PacketScheduler::HandleScenarioA()
    - Manages queues and predicts packet delays when there are packets in queues that may not be transmitted in time, and the last packet of them is within the queue.
  - DL_PPM_PacketScheduler::HandleScenarioB()
    - Manages queues and predicts packet delays when there are packets in queues that may not be transmitted in time, but the last packet of them is <u>not</u> within the queue.
- **Phase III: Cut-in Process**
  - DL_PPM_PacketScheduler::AllocCutInUserZForPrbY()
    - Allocates a cut-in user *z* from all candidate users who will make the least decrease in throughputs among all others to use PRB *y*.

Lastly, the PRB allocation function is modified to handle the aforementioned functions for the PPM in each phase:

- DL_PPM_PacketScheduler::RBsAllocation()

The next three sub-sections will describe the three phases of the PPM downlink scheduling algorithm from [4] in detail and outline the implementation of the PPM. However, for the actual realization of the PPM downlink packet scheduler, please refer to the source code in the Appendix section.

2.3.3. PPM Phase I – Initial Scheduling for PRBs

This phase operates in the frequency domain. In order to achieve good throughputs, PRB *y* is allocated to user *x* with the best CQI among all other users by the following formula:

$$M_{x,y} = \arg\max_{x}\left(CQI_{x,y}\right),$$

This is implemented in **DL_PPM_PacketScheduler::AllocUserXWithMaxCqiToPrbY()**. For the CQI value of user *x* in PRB *y*, it can be obtained by calling **PacketScheduler::FlowToSchedule::GetCqiFeedbacks()**. Additionally, we add a new method, **PacketScheduler::FlowToSchedule::SetHasBestCqi()**, to mark the user that has $M_{x,y}$ as the one with the best CQI. We also add the corresponding accessor method, **PacketScheduler::FlowToSchedule::GetHasBestCqi()**, to allow Phase II to query whether users have good CQIs.

2.3.4. PPM Phase II – Queue Management and Packet Delay Prediction

Contrary to the previous phase, Phase II deals with users whose CQI values are not as good because they are possibly located at cell edges relative to eNBs. Their packets may not be transmitted to their destinations in time. If only considering throughputs, there will likely be many out-of-date packets that get discarded at destinations. Instead, packet types, delays and timestamps should be considered to satisfy the QoS requirements. Timestamps refer to instants that packets are generated at the application layer. Continuous packets within the same MAC queue may belong to the same type and have similar delay bounds and timestamps. Packet *i* in queue *k* for user *x* will not be transmitted in time if:

$$Db_{k,i} - \left(t_{ok} - t_{k,i}\right) < R_{k,i}T_I + t_{pi}\ .$$

When there are continuous packets that are not expected to be transmitted in time in the queue, there are two possible situations, as illustrated in the figure below. The queue spaces coloured in blue contain packets that are expected to be transmitted in time. Those coloured in red are out-of-date packets that are overdue. The white spaces are not occupied with any packets.

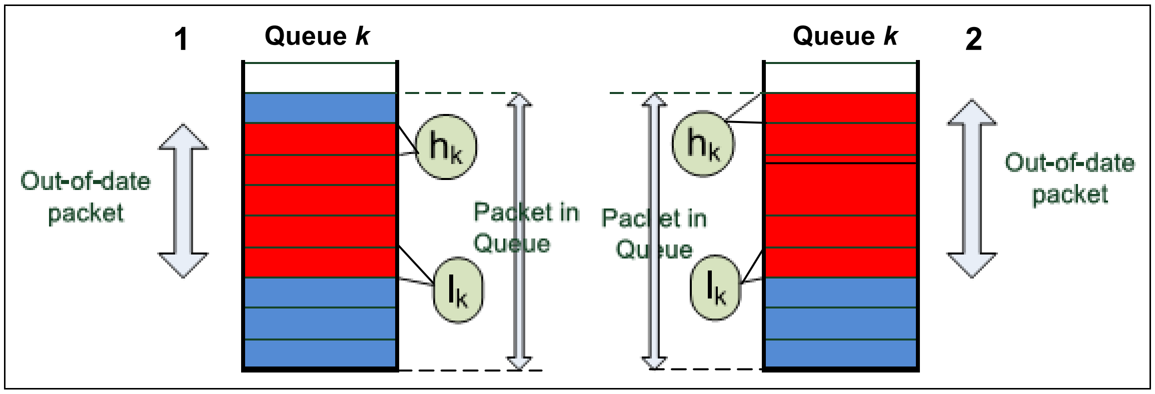

Figure 5: Scenario A – two situations for continuous out-of-date packets in the queue [4]

In situation 1, there are continuous out-of-date packets in the queue, but the last packet in the queue can be sent out in time. In situation 2, the difference is that the last packet in the queue is part of the continuous out-of-date packets and thus cannot be sent out in time. Both situations can be generalized as Scenario A and are to be handled in the function **DL_PPM_PacketScheduler::HandleScenarioA()** for user *x*:

Scenario A: Queue *k* contains packets that may not be transmitted in time, and the last packet of these overdue packets is within the queue.

The first and last overdue packets in queue *k* can be identified as follows:

$$l_k = \arg\min_i \left(Db_{k,i} - \left(t_{ok} - t_{k,i} \right) < R_{k,i} T_I + t_{pi} \right),$$

$$h_k = \arg\max_i \left(Db_{k,i} - \left(t_{ok} - t_{k,i} \right) < R_{k,i} T_I + t_{pi} \right).$$

The estimated packet loss rate is calculated by:

$$L_k = \frac{h_k - l_k + 1}{N_k + h_k}.$$

The QoS requirements for the packets in queue *k* cannot be met if L_k is greater than or equal to the pre-defined threshold, T_k, then the cut-in process in Phase III will be entered. Otherwise, those overdue packets in queue *k* will be discarded, as having L_k smaller than T_k means that the queue can tolerate a larger loss rate. Instead, the channel bandwidth can be allocated to other packets that can be transmitted in time. In this case, the cut-in process will not be called. To support erasing a range of packets from l_k to h_k in the queue, we add a new method called **MacQueue::DiscardPackets()** in LTE-Sim.

Now, based on the two situations mentioned earlier, if the last packet that cannot be transmitted in time is not in queue *k*, then another scenario as shown in Figure 6 arises, which is to be handled in the function **DL_PPM_PacketScheduler::HandleScenarioB()**:

Scenario B: Queue *k* contains packets that may not be transmitted in time, and the last packet of these overdue packets is not within the queue.

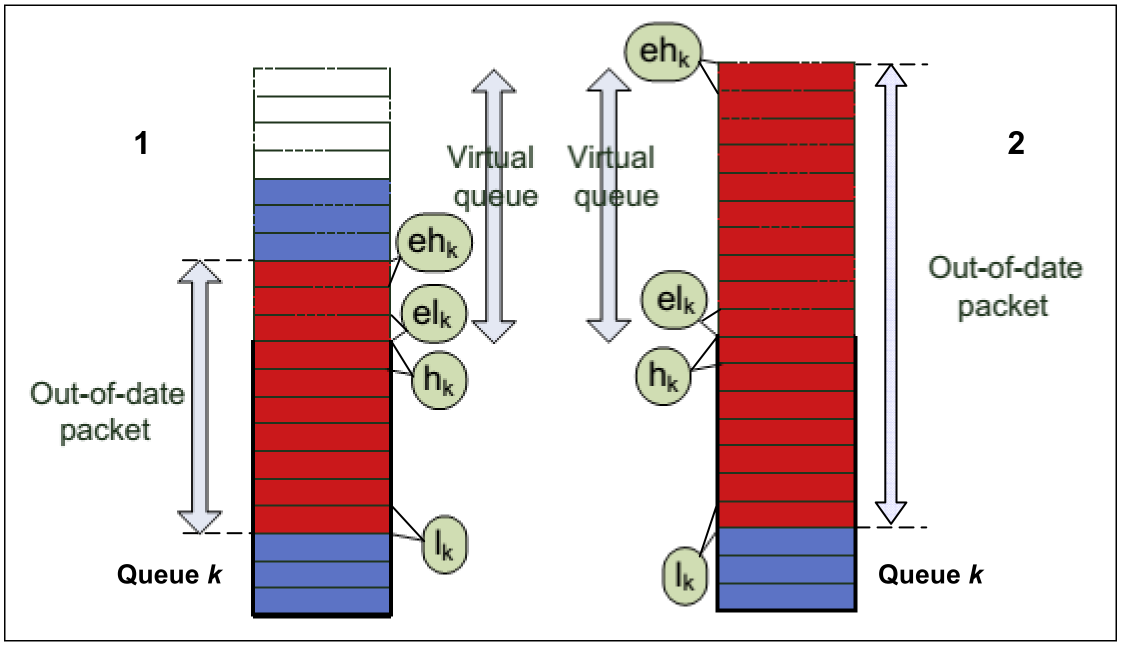

Figure 6: Scenario B – two situations for continuous out-of-date packets in the queue [4]

For packets that are not in queue *k*, a virtual queue will be used to store them. The goal is to predict the arrival and stamp times of future incoming packets in physical queue *k* and/or the virtual queue of it based on the inter-arrival and inter-stamp times of current packets. The predictions can determine whether new incoming packets can be transmitted in time. To find the positions of the first and last overdue packets in the virtual queue, the average inter-arrival and stamp times for the current packets in queue *k* needs to be calculated first:

$$\Delta T_k = \frac{1}{n_k} \sum_{i=0}^{n_k - 1} \left(t_{k,i+1} - t_{k,i} \right) = \frac{t_{k,n_k} - t_{k,0}}{n},$$

$$\Delta T_{sk} = \frac{1}{n_k} \sum_{i=0}^{n_k - 1} \left(t_{sk,i+1} - t_{sk,i} \right) = \frac{t_{sk,n_k} - t_{sk,0}}{n}$$

where n_k is the number of packets in queue *k*. The (n_k-1)-th packet, which is the last packet, will be transmitted in the following cycle:

$$R_{k,n_k - 1} = \left\lceil \frac{n_k}{EU_x T_I} \right\rceil.$$

Thus, the amount of time needed to transmit n_k packets in queue *k* is:

$$T_{n_k} = R_{k,n_k - 1} T_I .$$

This time and the average inter-arrival time calculated earlier allow us to estimate the number of new arrival packets into queue *k* during the eq_{n_k} interval that all current n_k packets in the queue are transmitted:

$$eq_{n_k} = \left\lceil \frac{T_{n_k}}{\Delta T_k} \right\rceil.$$

The estimated arrival and stamp times for packet *n* in the virtual queue for queue *k* are then:

$$et_{k,n} = t_{k,n_k-1} + (n+1)\Delta T_k \,,$$

$$et_{sk,n} = t_{sk,n_k-1} + (n+1)\Delta T_{sk} \,.$$

Please note that the counter '*n*' is used to denote the packet number in the virtual queue to differentiate it from the counter '*i*' used for the physical queue. Next, the interval for the *n*-th packet in virtual queue *k* to be transmitted can be estimated by:

$$ER_{k,n} = \left\lceil \frac{n_k + n + 1}{EU_x T_I} \right\rceil .$$

On the other hand, packet *n* will not be transmitted in time if:

$$Db_{k,n} - \left(t_{ok} - et_{k,n}\right) < ER_{k,n} T_I + t_{pn} \,.$$

In the end, the positions of the first and last packets that may become overdue in the virtual queue of queue *k* can be found by:

$$el_k = \arg\min_n \left(Db_{k,n} - \left(t_{ok} - et_{k,n}\right) < ER_{k,n} T_I + t_{pn}\right),$$

$$eh_k = \arg\max_n \left(Db_{k,n} - \left(t_{ok} - et_{k,n}\right) < ER_{k,n} T_I + t_{pn}\right).$$

The estimated packet loss rate is then calculated by the following formulas:

$$eL_k = \begin{cases} \dfrac{(h_k - l_k + 1)(eh_k - el_k + 1)}{N_k + h_k + eh_k} & \text{if } 0 \le (eh_k - el_k) < th_k \\ T_k + \varepsilon & \text{if } (eh_k - el_k) \ge th_k \\ \dfrac{(h_k - l_k + 1)}{N_k + h_k} & \text{if } el_k \text{ cannot be found} \end{cases} ,$$

where th_k is a threshold whose value can be set to be larger than the size of queue *k*, enabling us to evaluate if the packet arrival rate is greater than the departure rate, and ε is a small number that ensures the cut-in process can be entered to handle burst of packet losses in queue *k*. For simplicity, we set th_k to be **1** packet larger than the size of queue *k* for user *x* and ε to be **0.1**. For the case that eL_k cannot be found, it is when the packet in the last space of queue *k* is also the last overdue packet, the formula is the same as that of L_k, meaning that the out-of-date packets beyond the physical queue can be omitted.

If the calculated eL_k is greater than or equal to the threshold, T_k, the QoS requirements for the packets in physical/virtual queue *k* cannot be met, the cut-in process in Phase III will be invoked. Otherwise, the overdue packets are discarded by calling the newly added method, **MacQueue::DiscardPackets()**, since the current queue is able to accept a larger packet loss rate. This allows other in-time packets to use the channel bandwidth; the cut-in process will not be called in this case.

Finally, if all packets in queue *k* can be transmitted in time, the cut-in process in Phase III will not be entered. This scenario is known as:

Scenario C: All the packets contained in queue *k* will be transmitted in time.

Scenario C does not involve any queue management and delay prediction for packets; thus, we do not need to define a function in the **DL_PPM_PacketScheduler** class to handle this scenario. Finally, we wrap up **DL_PPM_PacketScheduler::HandleScenarioA()** and **DL_PPM_PacketScheduler::HandleScenarioB()** from Scenarios A and B by the single function, **DL_PPM_PacketScheduler::ManageQueuesAndPredictPktDelays()**.

2.3.5. PPM Phase III – Cut-in Process

Phase III is implemented in the function **AllocCutInUserZForPrbY()**. It will try to find a cut-in user *z* from all candidate users based on the following formula:

$$C_{z,y} = \min\left(ET_{x,y} - ET_{z,y}\right),$$

It iteratively searches for user *z* has the least decrease in throughputs until all cut-in users are processed or all PRBs are allocated. If all cut-in users have been handled, the remaining PRBs are simply allocated to users with maximum throughputs. If all PRBs have been allocated, those cut-in users need more PRBs than the PRBs available. If the cut-in process is not called, PRB *y* is allocated to user *x* having the maximum throughput. In general, cut-in users with the highest throughputs are selected to utilize PRBs.

Chapter 3. Simulation

In order to evaluate the effectiveness of the PPM algorithm implemented, we use LTE‑Sim and compare the PPM results with the ones produced by the PF, MLWDF, and EXPPF downlink scheduling schemes by simulating transmission of a real-time video stream from the eNB in the central cell to several UEs within the same cell. The scenario, as illustrated in Figure 7, is known as "single-cell multi-user with interference" in LTE-Sim, which is defined in /lte-sim/src/scenarios/single-cell-with-interference.h. It considers the possible influence of nearby eNBs that can generate radio interference by creating other eNBs in the surrounding neighbouring cells in addition to the eNB in the central cell. This scenario is more realistic to the nature of a real-life LTE communication environment and yet not so complicated. The aforementioned header file has cases to select the PF, MLWDF, and EXPPF downlink schedulers; we need to modify it to be able to support the use of the PPM.

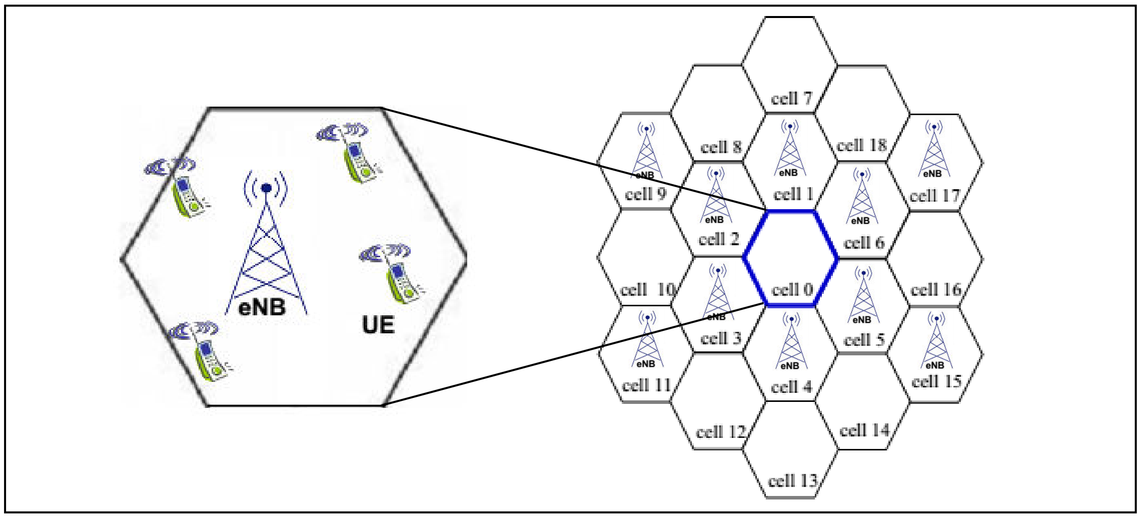

Figure 7: LTE-Sim Single Cell with Interference Scenario

3.1. Simulation Parameters

The following table presents the various parameters we use for our simulations. They are mostly chosen to be identical to the ones used by the PPM paper in [4] so we are able to demonstrate that our implemented PPM downlink scheduler in LTE-Sim works consistently. For other parameters not explicitly mentioned, they are the default values in the simulator.

Number of cells	10
Minimum number of UEs	10
Interval between UEs	10
Maximum number of UEs	40
Number of eNBs	1 in each cell
Transmission radius of eNBs	3 km
Downlink Bandwidth	20 MHz
Number of PRBs	100 (12 sub-carriers per PRB)
Modulation and coding schemes	QPSK, 16 QAM, 64, QAM
Number of video flows	1 real-time H.264 encoded video from /lte-sim/src/flows/application/Trace/foreman_H264_128k.dat
Video bit rate	128 kbps
Traffic generator	Trace-based
Downlink schedulers	PF, MLWDF, EXP, and PPM (simulated in this order)
Frame structure	FDD
Speed of UEs	30 km/h
Mobility model of UEs	Random direction
Delay bound	0.5 s
Simulation duration	180 s
Flow duration	120 s

Table 3: LTE-Sim Simulation Parameters

3.2. Simulation Results

We simulate transmitting the 128 kbps real-time video stream from the eNB to all the UEs, using LTE-Sim. Each UE moves at 30 km/h within the central cell, which resembles the average speed between the pedestrian and vehicular scenarios. With the mobility model of random direction in use, the UEs will not move out of the simulation boundary of the eNB but will randomly choose their direction. Once they reach the boundary, they move towards another random direction. The next five sub-sections will compare the following perspectives between the PF, MLWDF, EXPPF, and PPM downlink schedulers:

- **Throughput (in Megabits per second, Mbps)**
  - This is the total application-level throughput for all UEs, which is the number of useful information bits delivered from the eNB to all UEs within the central cell. It only focuses on the successful receipt of packets at the destination. Those packets that have errors or fail to arrive in time do not meet their QoS requirements and thus are not counted.
- **Delay (in seconds, s)**
  - This measures the amount of time it takes for the packets travel across the LTE network from the eNB to all UEs. Only those packets that successfully reach the destination are counted.
- **Fairness Index**
  - This determines whether the UEs are receiving a fair share of resources to meet the QoS requirements. If a downlink scheduler does not consider the fairness aspect, UEs are generally allocated with more resources when they are close to the eNB, and edge UEs usually suffer from resource starvation. The fairness measure used in LTE-Sim is Jain's fairness index, which is calculated by the following equation:

$$\Gamma(x_1, x_x, \ldots, x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \sum_{i=1}^{n} x_i^{\;2}},$$

where x_i is the application-level throughput for the *i*-th connection, and *n* is the number of UEs. Γ ranges from the worst case, 1/*n*, to the best case, 1, which are achieved when only one and all UEs receive(s) the allocation, respectively. In other words, the index value is *k*/*n* when *k* UEs share the resources equally, whereas the other (*n* - *k*) UEs receive nothing.

- **Packet Loss Ratio (PLR)**
  - This measures the ratio of the amount of packet losses at the receiving end, which is calculated by:

$$PLR = \frac{\text{Number of packets sent - Number of packets received}}{\text{Number of packets sent}}.$$

A good downlink scheduling scheme should yield a low PLR.

- **Spectral Efficiency**
  - This refers to the application-level information rate in bits per second that can be transmitted over the specified downlink bandwidth, which is:

$$\text{Spectral Efficiency} = \frac{\text{Throughput achieved by all UEs}}{\text{Bandwidth}}.$$

The higher the spectral efficiency, the more efficient that a frequency spectrum can be utilized by UEs' channel access.

3.2.1. Throughput

The following graph shows the application-level throughputs versus the number of UEs. As can be seen, due to taking account of both throughputs, managing possible overdue packets in the queues, and predicting delays for future incoming packets, the PPM achieves the best results among all other downlink schedulers, especially when the number of UEs increases. For other scheduling schemes, MLWDF and EXPPF have the similar results since they both consider the HOL delays. PF yields the worst throughputs because it allocates flows to use PRBs solely based on weights.

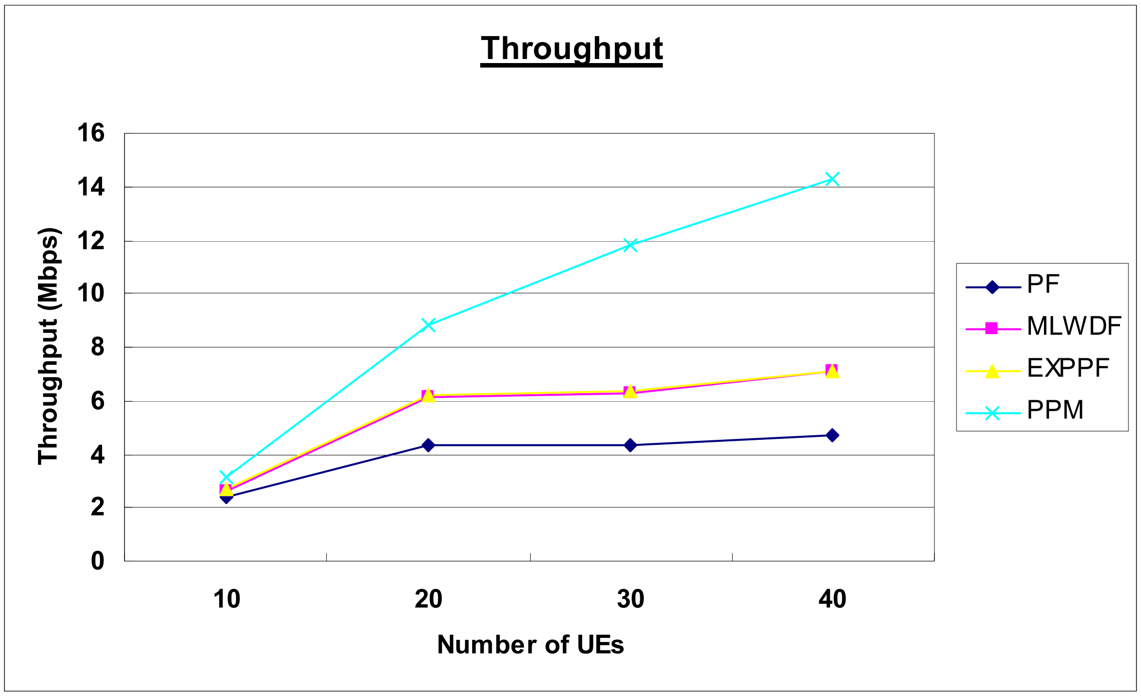

Figure 8: Throughput Comparison between PF, MLWDF, EXPPF, and PPM

Comparing our simulation results for PF, MLWDF, EXPPF, and PPM in Figure 8 with the results from the PPM paper in [4] shown in the following figure, their throughput values for the cases of 30 and 40 UEs (circled in red) are very similar. Hence, we can say that our implemented PPM downlink scheduler functions well as expected.

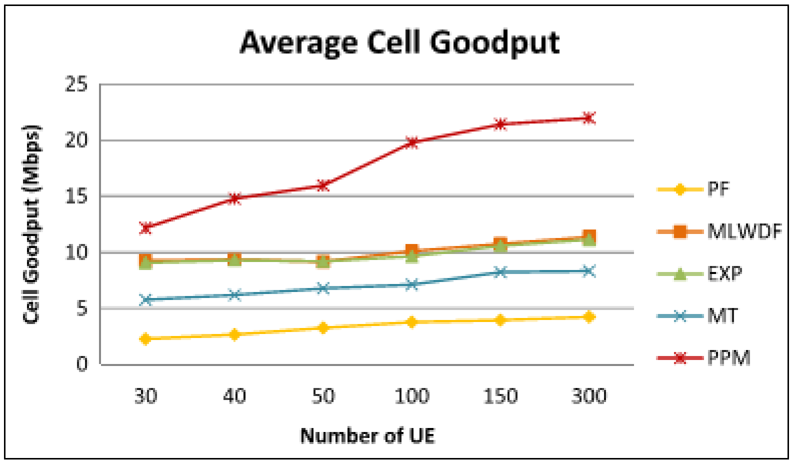

Figure 9: Average Cell Goodput (UE speed = 30 km/h) from the PPM paper [4]

3.2.2. Delay

Figure 10 shows the delay time versus the number of UEs. The PPM has the lowest delay among all other downlink schedulers. By the similar reasoning as the throughput perspective, MLWDF and EXPPF have the similar second best results since they both consider the HOL delays. PF yields the highest delay because it uses weights to determine which of the packet flows to transmit. Since all MLWDF, EXPPF, and PPM discard out-of-date packets as part of their scheduling algorithms, their delays are all pretty low.

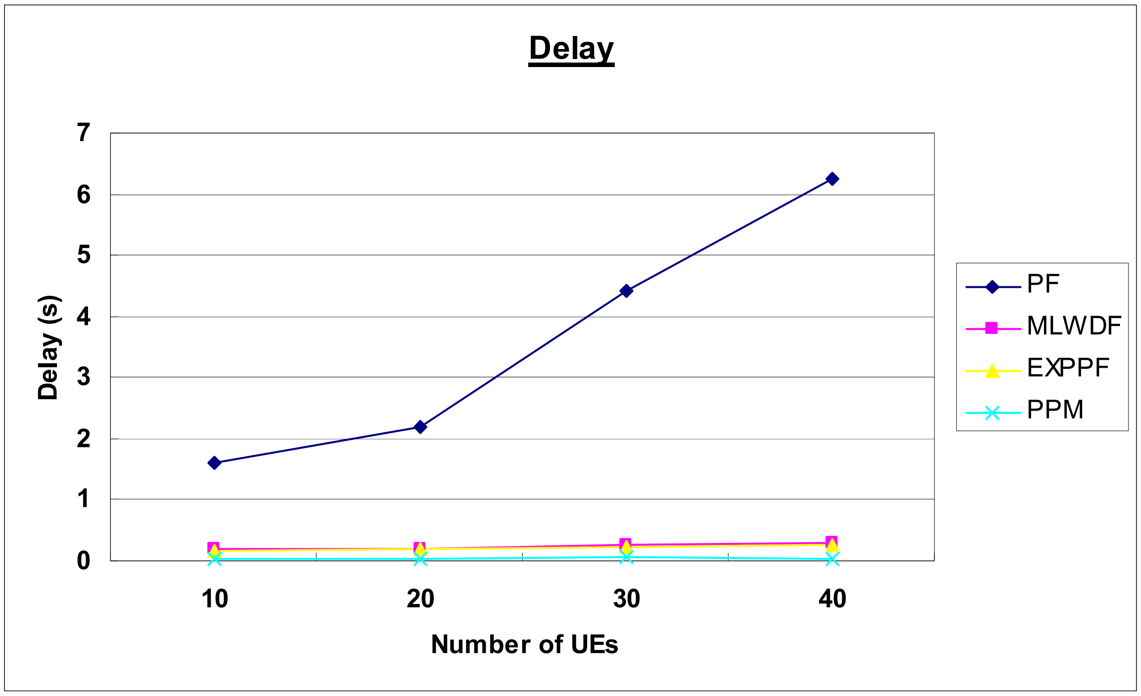

Figure 10: Delay Comparison between PF, MLWDF, EXPPF, and PPM

Next, comparing our results for PF, MLWDF, EXPPF, and PPM in Figure 10 with the ones from the PPM paper in [4] shown below, their delay values for the cases of 30 and 40 UEs (circled in red) are similar. Thus, our implemented PPM in LTE-Sim works as expected.

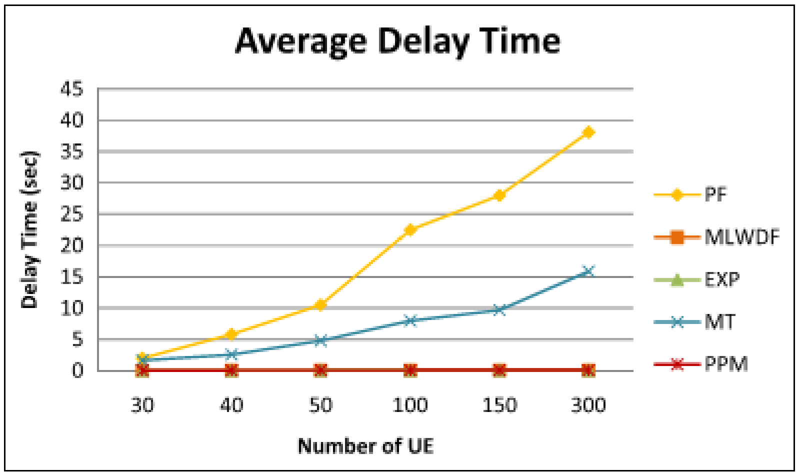

Figure 11: Average delay time (UE speed = 30 km/h) [4]

3.2.3. Fairness Index

The following curves present the fairness index versus the number of UEs. The PPM has the highest values among all other downlink schedulers, meaning that this algorithm provides a better level of fairness than other schemes. The results for MLWDF and EXPPF are again similar, and PF has the worst performance. The trend shows that the fairness index value starts to decline when the number of UEs increases. This phenomenon is expected since more UEs are competing for the limited resources.

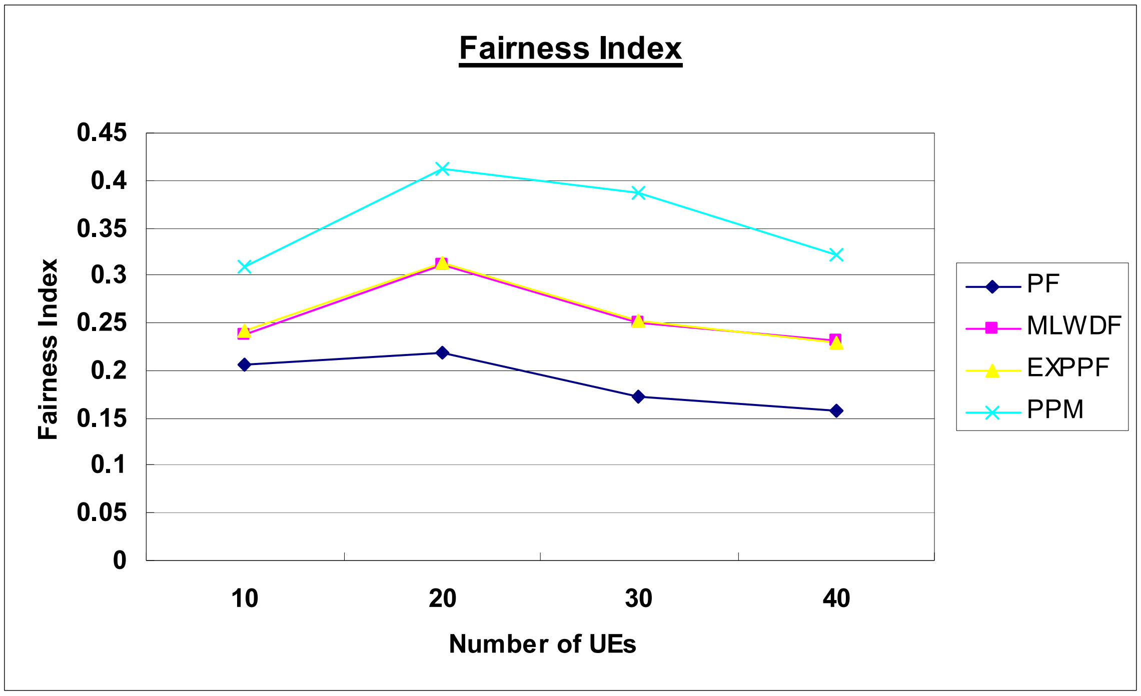

Figure 12: Fairness Index Comparison between PF, MLWDF, EXPPF, and PPM

3.2.4. Packet Loss Ratio

The following figure shows the PLR experienced by the video flows. The PLR increases with the number of UEs because the number of concurrent real-time flows will be larger, implying that the tendency for the downlink scheduler to discard overdue packets will be higher. Nevertheless, the PPM still gives the lowest PLR among all other schemes. MLWDF and EXPPF have the similar second best results again, and PF has the worst PLR. The reason is that PPM, "MLWDF, and EXPPF initiate their prevention mechanisms when the number of users increases and drop packets which cannot reach their destinations in time" [4], where as PF never drops any packets. Hence, PPM's prediction of the behaviour of future incoming packets based on current packets in the queue is helpful.

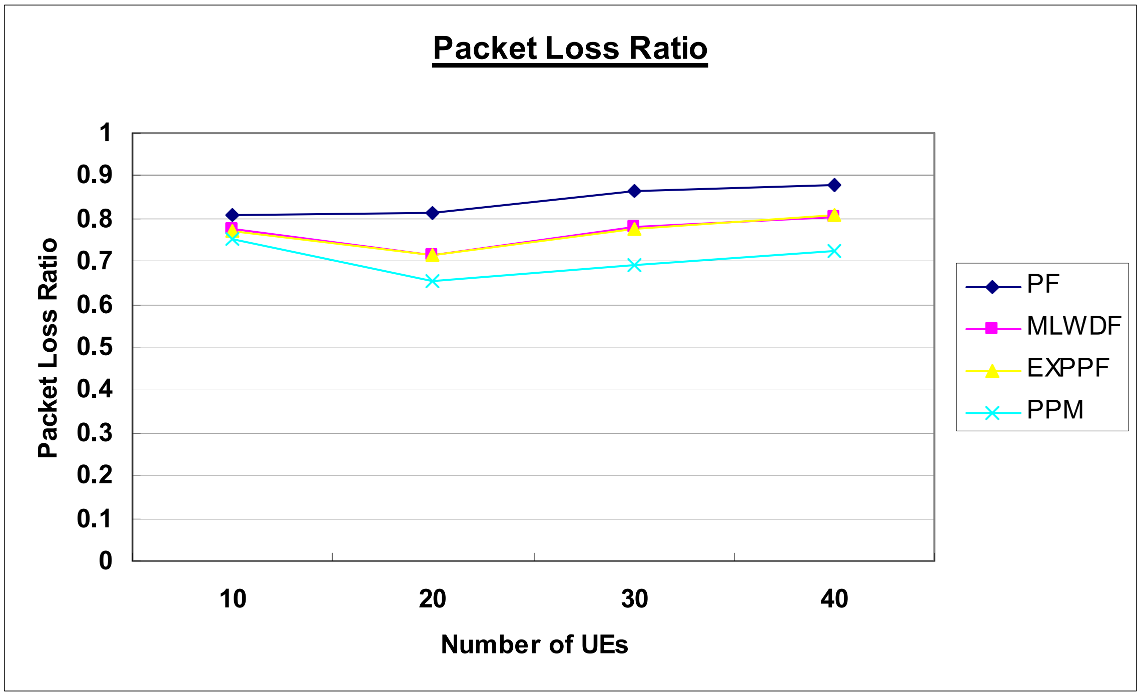

Figure 13: Packet Loss Ratio Comparison between PF, MLWDF, EXPPF, and PPM

3.2.5. Spectral Efficiency

Finally, Figure 14 shows the spectral efficiency curves versus the number of UEs. Again, the PPM has the highest values among all other downlink schedulers even when the number of UEs increases. The second best one is PF; it performs better than that of MLWDF and EXPPF this time. The reason is that “the QoS-aware schedulers like MLWDF and EXPPF still try to guarantee QoS constraints to a high number of flows, with a consequent negative impact on the system efficiency” [5].

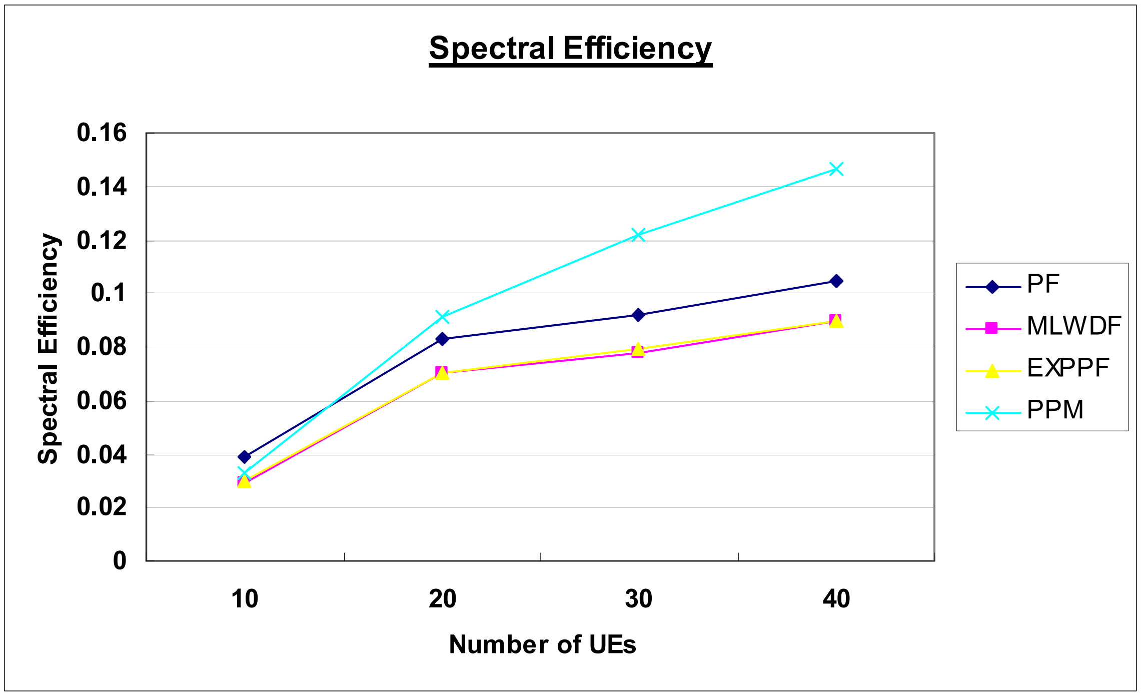

Figure 14: Spectral Efficiency Comparison between PF, MLWDF, EXPPF, and PPM

Chapter 4. Conclusion

In this project, we have successfully implemented the Packet Prediction Mechanism (PPM) downlink scheduler in LTE-Sim, and evaluated its performance through various simulations. This QoS-aware scheduling algorithm for real-time services in LTE networks consists of three phases. Phase I operates in the frequency domain, allowing a good application-level throughput to be achieved by selecting flows that have best Channel Quality Indication (CQI) indices. Phase II in the time domain predicts packet delays and loss rates for future incoming packets based on the behaviour of current packets in the physical queue with the use of a virtual queue when necessary. Using the predicted results from the previous phase, Phase III then employs a cut-in process for rearranging the transmission order and discarding packets that cannot meet their delay requirements. Since the PPM scheduling scheme consider both throughputs in the frequency domain and delay times in the time domain, the LTE-Sim simulation results have demonstrated that the PPM out-performs the Priority First (PF), Modified Largest Weighted Delay First (MLWDF), and Exponential PF (EXPPF) downlink schedulers in terms of Throughput, Delay, Fairness Index, Packet Loss Ratio (PLR), and Spectral Efficiency.

References

[1] White Paper “LTE: An Introduction,” Ericsson AB, 2011. <http://www.ericsson.com/res/docs/2011/lte_an_introduction.pdf>.

[2] A. Khan, “LTE Network Infrastructure and Elements,” LTE Encyclopedia, Oct. 2011, <https://sites.google.com/site/lteencyclopedia>.

[3] N. Wiffen, “Understanding 3GPP Bearer,” LTE/HSPA/EPC Knowledge Explained, Red Banana Wireless Ltd., 2013, <http://www.red-banana.org>.

[4] W. Lai, C. Tang, “QoS-aware Downlink Packet Scheduling for LTE Networks,” Computer Networks: The International Journal of Computer and Telecommunications Networking, vol. 57 issue 7, pp. 1689-1698, May 2013.

[5] G. Piro, L. A. Grieco, G. Boggia, F. Capozzi, and P. Camarda, “Simulating LTE Cellular Systems: an Open Source Framework,” IEEE Trans. Veh. Technol., vol. 60, no. 2, Feb. 2011, <http://telematics.poliba.it/index.php/en/lte-sim>.

[6] J.-G. Choi and S. Bahk, “Cell-throughput analysis of the proportional fair scheduler in the single-cell environment,” IEEE Trans. Veh. Technol., vol. 56, no. 2, pp. 766-778, Mar. 2007.

[7] F. Capozzi, G. Piro, L. A. Grieco, G. Boggia, and P. Camarda, “Downlink Packet Scheduling in LTE Cellular Networks: Key Design Issues and a Survey,” IEEE Commun. Surveys and Tutorials, vol. 15, no. 2, pp. 678-700, Apr. 2013.

Appendix A.

LTE-Sim Source Code

PPM Downlink Scheduling Algorithm Implemented

dl-ppm-packet-scheduler.cpp

```
 /*
 *-----------------------------------------------------------------------------
 *
 * dl-ppm-packet-scheduler.cpp
 *
 * The LTE-Sim downlink packet scheduler that implements the Packet Prediction
 * Mechanism (PPM) downlink scheduling algorithm proposed in this paper:
 *
 * "QoS-aware Downlink Packet Scheduling for LTE Networks,"
 * Computer Networks: The International Journal of Computer and
 * Telecommunications Networking archive, Volume 57 Issue 7, May, 2013,
 * Pages 1689-1698
 *
 * Howard Chang
 * Simon Fraser University
 * 20007-2192
 *
 *-----------------------------------------------------------------------------
 */


#include "dl-ppm-packet-scheduler.h"
#include "../mac-entity.h"
#include "../../packet/Packet.h"
#include "../../packet/packet-burst.h"
#include "../../../core/spectrum/bandwidth-manager.h"
#include "../../../device/ENodeB.h"
#include "../../../device/NetworkNode.h"
#include "../../../flows/MacQueue.h"
#include "../../../flows/QoS/QoSForPPM.h"
#include "../../../flows/application/Application.h"
#include "../../../flows/radio-bearer.h"
#include "../../../phy/lte-phy.h"
#include "../../../protocolStack/mac/AMCModule.h"
#include "../../../protocolStack/rrc/rrc-entity.h"
#include "../../../utility/eesm-effective-sinr.h"


//#define SCHEDULER_DEBUG_PPM
#define SCHEDULER_CHECKPOINT() \
    do { \
        if (m_showCheckpoint) { \
            std::cout << "CHECKPOINT - " << \
                __FUNCTION__ << \
                " (" << __LINE__ << ")" << \
                std::endl; \
        } \
    } while (0);
```

```
/* Transmission Time Interval (TTI) */
#define TTI     (0.001)   /* 1ms */
#define T_I     (1 * TTI)

/* Pre-defined threshold for the lost rate of packets in queue k */
#define T_Q_k   (0.5)       /* 0.5s for video application */


DL_PPM_PacketScheduler::DL_PPM_PacketScheduler ()
{
    m_showCheckpoint = false;
    SetMacEntity(0);
    CreateFlowsToSchedule();
}


DL_PPM_PacketScheduler::~DL_PPM_PacketScheduler ()
{
    Destroy();
}


void
DL_PPM_PacketScheduler::DoSchedule ()
{
#ifdef SCHEDULER_DEBUG
    std::cout <<
        "Start PPM DL packet scheduler for node " <<
        GetMacEntity()->GetDevice()->GetIDNetworkNode() <<
        std::endl;
#endif

    UpdateAverageTransmissionRate();

    /*
     * Check and drop packets at the MAC layer (i.e. those packets whose delay
     * exceed the maximum allowable value)
     */
    CheckForDLDropPackets();

    /* Select all flows that can be scheduled */
    SelectFlowsToSchedule();

    ComputeAverageOfHOLDelays();

    if (GetFlowsToSchedule()->size() == 0) {
        /* Do nothing */
    } else {
        RBsAllocation();
    }

    StopSchedule();
}


void
DL_PPM_PacketScheduler::RBsAllocation ()
{
#ifdef SCHEDULER_DEBUG_PPM
    std::cout << " ---- RBs Allocation (PPM): " << std::endl;
#endif
```

```
    /*
     * Get the list of downlink flows that have packets to transmit and can be
     * scheduled in the current sub-frame. These flows were created by the eNB.
     * The MAC queue length and CQI feedbacks are stored for each flow.
     */
    FlowsToSchedule *flows = GetFlowsToSchedule();
    int nbOfRBs = GetMacEntity()->GetDevice()->GetPhy()->GetBandwidthManager()-
>GetDlSubChannels().size();

    /* RBs allocation */
    for (int PRB_y = 0; PRB_y < nbOfRBs; PRB_y++) {
        AllocUserXWithMaxCqiToPrbY(PRB_y);
    }
    for (int PRB_y = 0; PRB_y < nbOfRBs; PRB_y++) {
        ManageQueuesAndPredictPktDelays(PRB_y);
    }

    /* Finalize the allocation */
    AMCModule *amc = GetMacEntity()->GetAmcModule();
    PdcchMapIdealControlMessage *pdcchMsg = new PdcchMapIdealControlMessage();

    for  (FlowsToSchedule::iterator  it  =  flows->begin();  it  !=  flows->end();
it++) {
        FlowToSchedule *flow = (*it);
        if (flow->GetListOfAllocatedRBs()->size () > 0) {
            /* This flow has been scheduled */
            std::vector<double> estimatedSinrValues;
            for  (unsigned  int  rb  =  0;  rb  <  flow->GetListOfAllocatedRBs()-
>size(); rb++) {
                double sinr =
                    amc->GetSinrFromCQI(
                        flow->GetCqiFeedbacks().at(flow-
>GetListOfAllocatedRBs()->at(rb)));

                estimatedSinrValues.push_back(sinr);
            }

            /* Compute the effective SINR */
            double effectiveSinr = GetEesmEffectiveSinr(estimatedSinrValues);

            /* Get the MCS for transmission */
            int mcs = amc->GetMCSFromCQI(amc->GetCQIFromSinr(effectiveSinr));

            /* Define the amount of bytes to transmit */
            int transportBlockSize = amc->GetTBSizeFromMCS(mcs);
            double bitsToTransmit =
                transportBlockSize * flow->GetListOfAllocatedRBs()->size();
            flow->UpdateAllocatedBits(bitsToTransmit);

#ifdef SCHEDULER_DEBUG
            std::cout <<
                "\t\t --> flow " <<
                flow->GetBearer()->GetApplication()->GetApplicationID() <<
                " has been scheduled: " <<
                "\n\t\t\t nb of RBs " <<
                flow->GetListOfAllocatedRBs()->size() <<
                "\n\t\t\t effectiveSinr " <<
                effectiveSinr <<
                "\n\t\t\t tbs " <<
                transportBlockSize <<
                "\n\t\t\t bitsToTransmit " <<
                bitsToTransmit <<
                std::endl;
#endif

            /* Create PDCCH messages */
            for  (unsigned  int  rb  =  0;  rb  <  flow->GetListOfAllocatedRBs()-
>size(); rb++) {
                pdcchMsg->AddNewRecord(
                    PdcchMapIdealControlMessage::DOWNLINK,
                    flow->GetListOfAllocatedRBs()->at(rb),
                    flow->GetBearer()->GetDestination(),
                    mcs);
            }
        }
    }

    if (pdcchMsg->GetMessage()->size() > 0) {
        GetMacEntity()->GetDevice()->GetPhy()-
>SendIdealControlMessage(pdcchMsg);
    }
    delete pdcchMsg;
}


/*
 ********************************************************************************
 * Phase 1: Initial scheduling for PRBs
 ********************************************************************************
 */
void
DL_PPM_PacketScheduler::AllocUserXWithMaxCqiToPrbY (int PRB_y)
{
    FlowsToSchedule *flows = GetFlowsToSchedule();
    int num_usrs = flows->size();

    /* CQI value if PRB y is allocated to user x */
    int CQI_usr_x_PRB_y = 0;
    int max_CQI_usr_x_PRB_y = 0;

    /* Maximum CQI_user_x_PRB_y for PRB y */
    int M_usr_x_PRB_y = -1;

    SCHEDULER_CHECKPOINT();

    for (int usr_x = 0; usr_x < num_usrs; usr_x++) {
        CQI_usr_x_PRB_y = flows->at(usr_x)->GetCqiFeedbacks().at(PRB_y);

        if (CQI_usr_x_PRB_y > max_CQI_usr_x_PRB_y) {
            max_CQI_usr_x_PRB_y = CQI_usr_x_PRB_y;
            M_usr_x_PRB_y = usr_x;
        }
    }

#ifdef SCHEDULER_DEBUG_PPM
    std::cout <<
        "\tPhase 1: PRB_y=" <<
        PRB_y <<
        ", num_usrs=" <<
        num_usrs <<
        ", M_usr_x_PRB_y=" <<
        M_usr_x_PRB_y <<
        ", max_CQI_usr_x_PRB_y=" <<
        max_CQI_usr_x_PRB_y <<
        ", ApplicationID=" <<
        flows->at(M_usr_x_PRB_y)->GetBearer()->GetApplication()-
>GetApplicationID() <<
        std::endl;
#endif

    if (M_usr_x_PRB_y >= 0) {
        /* The y-th RB has been allocated to this user x */
        flows->at(M_usr_x_PRB_y)->GetListOfAllocatedRBs()->push_back(PRB_y);

        /* Mark the user as the one with the best CQI */
        flows->at(M_usr_x_PRB_y)->SetHasBestCqi(true);
    }

    SCHEDULER_CHECKPOINT();
}


/*
 ******************************************************************************
 * Phase 2: Managing queues and prediction
 ******************************************************************************
 */
void
DL_PPM_PacketScheduler::ManageQueuesAndPredictPktDelays (int PRB_y)
{
    FlowsToSchedule *flows = GetFlowsToSchedule();
    int num_usrs = flows->size();
    double dl_bandwidth =
        GetMacEntity()->GetDevice()->GetPhy()->GetBandwidthManager()-
>GetDlBandwidth();

    /*
     * First identified packet number in queue k whose scheduled transmission
     * time may be late and in consequence the packet does not arrive at the
     * destination in time
     */
    int l_Q_k = -1;

    /*
     * Last identified packet number in queue k (if it exists) whose scheduled
     * transmission time may be late and in consequence the packet does not
     * arrive at the destination in time
     */
    int h_Q_k = -1;

    for (int usr_x = 0; usr_x < num_usrs; usr_x++) {
        FlowToSchedule *flow_usr_x = flows->at(usr_x);

        /* Only for user x whose CQI value is not as good */
        if (flow_usr_x->GetHasBestCqi() == false) {
            HandleScenarioA(
                usr_x,
                PRB_y,
                dl_bandwidth,
                l_Q_k,
                h_Q_k);
            SCHEDULER_CHECKPOINT();
            HandleScenarioB(
                usr_x,
                PRB_y,
                dl_bandwidth,
                l_Q_k,
                h_Q_k);
        }
    }
}


/*
 * Scenario A: There are packets in the queue which may not be transmitted in
 *             time, and the last packet which is not transmitted in time is
 *             within queue k.
 */
void
DL_PPM_PacketScheduler::HandleScenarioA (
    int     usr_x,
    int     PRB_y,
    double  dl_bandwidth,
    int    &l_Q_k,
    int    &h_Q_k)
{
    FlowsToSchedule    *flows = GetFlowsToSchedule();
    FlowToSchedule     *flow_usr_x = flows->at(usr_x);
    RadioBearer        *bearer_usr_x = flow_usr_x->GetBearer();
    QoSForPPM                       *qos_usr_x  =  (QoSForPPM  *)bearer_usr_x-
>GetQoSParameters();
    MacQueue           *mac_Q_usr_x = bearer_usr_x->GetMacQueue();
    int                 num_usrs = flows->size();

    /* Number of packets in queue k */
    int                 num_pkts_Q_k = mac_Q_usr_x->GetNbDataPackets();

    SCHEDULER_CHECKPOINT();

#ifdef SCHEDULER_DEBUG_PPM
    std::cout <<
        "\tPhase 2: usr_x=" << usr_x <<
        ", PRB_y=" << PRB_y <<
        ", num_pkts_Q_k=" << num_pkts_Q_k;
#endif

    if ((num_pkts_Q_k == 0) ||
        (mac_Q_usr_x->IsEmpty())) {
#ifdef SCHEDULER_DEBUG_PPM
    std::cout << " --> empty (Scenario A)" << std::endl;
#endif
        return;
    }

    l_Q_k = num_pkts_Q_k-1;
    h_Q_k = 0;

    for (int pkt_i = 0; pkt_i < num_pkts_Q_k; pkt_i++) {
        if (mac_Q_usr_x->GetPacketQueue()->at(pkt_i).GetPacket() == NULL) {
            continue;
        }

        /* Delay bound for packet i in queue k */
        double Db_Q_k_pkt_i = qos_usr_x->GetMaxDelay();
        //std::cout  <<  std::endl  <<  "Db_Q_k_pkt_i="  <<  Db_Q_k_pkt_i  <<
std::endl;

        /*
         * Observation time to calculate which packets in queue k may not
         * arrive at destinations before delay budgets
         */
        double to_Q_k = mac_Q_usr_x->GetTimeObservationForMayNotArrivePackets();
        //std::cout << "to_Q_k=" << to_Q_k << std::endl;

        /* Arrival time for the i-th packet in queue k */
        double ta_Q_k_pkt_i =
            mac_Q_usr_x->GetPacketQueue()->at(pkt_i).GetTimeArrival();
        //std::cout << "ta_Q_k_pkt_i=" << ta_Q_k_pkt_i << std::endl;

        /* Propagation delay in physical layer for packet i */
        double tp_pkt_i =
            mac_Q_usr_x->GetPacketQueue()->at(pkt_i).GetSize()     *     8     /
dl_bandwidth / 10e3;
        //std::cout << "tp_pkt_i=" << tp_pkt_i << std::endl;

        /* Average throughput for packets of user x in queue k */
        double E_Q_k_usr_x = bearer_usr_x->GetAverageTransmissionRate();
        //std::cout << "E_Q_k_usr_x=" << E_Q_k_usr_x << std::endl;

        /*
         * Interval that the i-th packet in queue k will be scheduled to
         * transmit at
         */
        double R_Q_k_pkt_i = ceil((pkt_i + 1) / (E_Q_k_usr_x * T_I));
        //std::cout << "R_Q_k_pkt_i=" << R_Q_k_pkt_i << std::endl;

        if ((Db_Q_k_pkt_i - (to_Q_k - ta_Q_k_pkt_i)) <
                (R_Q_k_pkt_i * T_I + tp_pkt_i)) {
            /* Packet i will not be transmitted in time */
            if (pkt_i < l_Q_k) {
                /*
                 * Update the number for the first packet which may
                 * become overdue in queue k
                 */
                l_Q_k = pkt_i;
            }
            if (pkt_i > h_Q_k) {
                /*
                 * Update the number for the last packet which may
                 * become overdue in queue k
                 */
                h_Q_k = pkt_i;
            }
            //std::cout  <<  "IN:  l_Q_k="  <<  l_Q_k  <<  ",  h_Q_k="  <<  h_Q_k  <<
std::endl;
        }
    }

    //std::cout << "OUT: l_Q_k=" << l_Q_k << ", h_Q_k=" << h_Q_k << std::endl;

    if (l_Q_k > h_Q_k) {
#ifdef SCHEDULER_DEBUG_PPM
    std::cout << " --> no overdue (Scenario A -> C)" << std::endl;
#endif
```

```
        return;
    }

    /*
     * Number of continuously successful transmission packets in queue k before
     * the current observation time and after the last dropped packet
     */
    int N_Q_k = mac_Q_usr_x->GetNbOfTxPacketsBeforeTimeObservation();
    //std::cout << "N_Q_k=" << N_Q_k << std::endl;

    /* Estimated loss rate for queue k */
    double L_Q_k = (h_Q_k - l_Q_k + 1) / (N_Q_k + h_Q_k);
    //std::cout << "L_Q_k=" << L_Q_k << std::endl;

#ifdef SCHEDULER_DEBUG_PPM
    std::cout <<
        ", l_Q_k=" << l_Q_k <<
        ", h_Q_k=" << h_Q_k <<
        ", L_Q_k=" << L_Q_k <<
        ", T_Q_k=" << T_Q_k;
#endif

    if (l_Q_k == h_Q_k) {
#ifdef SCHEDULER_DEBUG_PPM
        std::cout << " --> all in time (Scenario A -> C)" << std::endl;
#endif
    } else if (L_Q_k >= T_Q_k) {
        /*
         * QoS for packets in queue k cannot be met.
         * Call the cut-in process.
         */
#ifdef SCHEDULER_DEBUG_PPM
        std::cout << " --> cut-in (Scenario A)" << std::endl;
#endif
        AllocCutInUserZForPrbY(usr_x, PRB_y);
    } else {
        /*
         * Discard the overdue packets in queue k.
         * Do not call the cut-in process.
         */
#ifdef SCHEDULER_DEBUG_PPM
        std::cout << " --> discard overdue (Scenario A)" << std::endl;
#endif
        mac_Q_usr_x->DiscardPackets(l_Q_k, h_Q_k);
    }

    SCHEDULER_CHECKPOINT();
}


/*
 * Scenario B: There are packets in the queue which may not be transmitted in
 *             time, and the last packet which is not transmitted in time is
 *             not within queue k.
 */
void
DL_PPM_PacketScheduler::HandleScenarioB (
    int     usr_x,
    int     PRB_y,
    double  dl_bandwidth,
    int     l_Q_k,
    int     h_Q_k)
{
    FlowsToSchedule    *flows = GetFlowsToSchedule();
    RadioBearer        *bearer_usr_x = flows->at(usr_x)->GetBearer();
    QoSForPPM                        *qos_usr_x   =   (QoSForPPM   *)bearer_usr_x-
>GetQoSParameters();
    MacQueue           *mac_Q_usr_x = bearer_usr_x->GetMacQueue();
    int                 num_pkts_Q_k = mac_Q_usr_x->GetNbDataPackets();

    /* Average inter-arrival time for packets in queue k */
    double              delta_Ta_Q_k = 0;
    double              total_Ta_Q_k = 0;
    double              ta_Q_k_pkt_next = 0;
    double              ta_Q_k_pkt_last = 0;

    /* Average inter-stamp time for packets in queue k */
    double              delta_Ts_Q_k = 0;
    double              total_Ts_Q_k = 0;
    double              ts_Q_k_pkt_next = 0;
    double              ts_Q_k_pkt_last = 0;
    double                                     E_Q_k_usr_x   =   bearer_usr_x-
>GetAverageTransmissionRate();

    SCHEDULER_CHECKPOINT();

#ifdef SCHEDULER_DEBUG_PPM
    std::cout <<
        "\tPhase 2: usr_x=" << usr_x <<
        ", PRB_y=" << PRB_y <<
        ", num_pkts_Q_k=" << num_pkts_Q_k;
#endif

    if ((num_pkts_Q_k == 0) ||
        (mac_Q_usr_x->IsEmpty())) {
#ifdef SCHEDULER_DEBUG_PPM
    std::cout << " --> empty (Scenario B)" << std::endl;
#endif
        return;
    }

    if (l_Q_k > h_Q_k) {
#ifdef SCHEDULER_DEBUG_PPM
    std::cout << " --> no overdue (Scenario B -> C)" << std::endl;
#endif
        return;
    }

    /* For queue k */
    for (int pkt_i = 0; pkt_i < num_pkts_Q_k-1; pkt_i++) {
        if ((mac_Q_usr_x->GetPacketQueue()->at(pkt_i).GetPacket() == NULL) ||
            (mac_Q_usr_x->GetPacketQueue()->at(pkt_i+1).GetPacket() == NULL)) {
            continue;
        }

        /* Calculate the total inter-arrival time for current packets in queue
k. */
        ta_Q_k_pkt_next =
            mac_Q_usr_x->GetPacketQueue()->at(pkt_i+1).GetTimeArrival();
        double ta_Q_k_pkt_i =
            mac_Q_usr_x->GetPacketQueue()->at(pkt_i).GetTimeArrival();
        total_Ta_Q_k += (ta_Q_k_pkt_next - ta_Q_k_pkt_i);
        /* Calculate the total inter-stamp time for current packets in queue k.
*/
        ts_Q_k_pkt_next =
            mac_Q_usr_x->GetPacketQueue()->at(pkt_i+1).GetTimeStamp();
        /* Stamp time for the i-th packet in queue k */
        double ts_Q_k_pkt_i =
            mac_Q_usr_x->GetPacketQueue()->at(pkt_i).GetTimeStamp();
        total_Ts_Q_k += (ts_Q_k_pkt_next - ts_Q_k_pkt_i);

        /* Get the last arrival and stamp times for current packets in queue k.
*/
        if (pkt_i == num_pkts_Q_k-2) {
            ta_Q_k_pkt_last = ta_Q_k_pkt_i;
            ts_Q_k_pkt_last = ts_Q_k_pkt_i;
        }
    }

    SCHEDULER_CHECKPOINT();

    /*
     * Calculate the average inter-arrival and inter-stamp times for current
     * packets in queue k.
     */
    delta_Ta_Q_k = total_Ta_Q_k / num_pkts_Q_k;
    delta_Ts_Q_k = total_Ts_Q_k / num_pkts_Q_k;

    /* Transmit the (num_pkts_Q_k-1)-th (last) packet in the following cycle.
*/
    double R_Q_k_pkt_last = ceil(num_pkts_Q_k / (E_Q_k_usr_x * T_I));

    /* Calculate the time needed to transmit num_pkts_Q_k packets in queue k.
*/
    double T_num_pkts_Q_k = R_Q_k_pkt_last * T_I;

    /*
     * Calculate the estimated number of new arrival packets into virtual
     * queue k during the time interval that all num_pkts_Q_k packets currently
     * in queue k are transmitted.
     */
    double eq_n_VQ_k = ceil(T_num_pkts_Q_k / delta_Ta_Q_k);

    /**********************************************************************/

    /*
     * First estimated packet number in the virtual queue k whose scheduled
     * transmission time may be late and in consequence the packet does not
     * arrive at the destination in time
     */
    int el_VQ_k = num_pkts_Q_k-1;

    /*
     * Last estimated packet number in the virtual queue k (if it exists) whose
     * scheduled transmission time may be late and cause the packet not to
     * arrive at the destination in time
     */
    int eh_VQ_k = 0;

    SCHEDULER_CHECKPOINT();

    for (int pkt_n = 0; pkt_n < num_pkts_Q_k; pkt_n++) {
        if (mac_Q_usr_x->GetPacketQueue()->at(pkt_n).GetPacket() == NULL) {
            continue;
        }

        /* Estimated arrival time for the n-th packet in virtual queue k */
        double eta_VQ_k_pkt_n = ta_Q_k_pkt_last + (pkt_n + 1) * delta_Ta_Q_k;

        /* Estimated stamp time for the n-th packet in virtual queue k */
        double ets_VQ_k_pkt_n = ts_Q_k_pkt_last + (pkt_n + 1) * delta_Ts_Q_k;

        /*
         * Estimated ER_n interval that the n-th packet in virtual queue k will
         * be scheduled to transmit at
         */
        double ER_VQ_k_pkt_n = ceil((num_pkts_Q_k + pkt_n + 1) / (E_Q_k_usr_x *
T_I));

        double Db_Q_k_pkt_n = qos_usr_x->GetMaxDelay();
        double to_Q_k = mac_Q_usr_x->GetTimeObservationForMayNotArrivePackets();

        /* Propagation delay in physical layer for packet n */
        double tp_pkt_n =
            mac_Q_usr_x->GetPacketQueue()->at(pkt_n).GetSize()     *     8     /
dl_bandwidth / 10e3;

        if ((Db_Q_k_pkt_n - (to_Q_k - eta_VQ_k_pkt_n)) <
                (ER_VQ_k_pkt_n * T_I + tp_pkt_n)) {
            /* Packet n will not be transmitted in time */
            if (pkt_n < el_VQ_k) {
                /*
                 * Update the number for the first packet which may become
                 * overdue in virtual queue k
                 */
                el_VQ_k = pkt_n;
            }
            if (pkt_n > eh_VQ_k) {
                /*
                 * Update the number for the last packet which may become
                 * overdue in virtual queue k
                 */
                eh_VQ_k = pkt_n;
            }
        }
    }

    SCHEDULER_CHECKPOINT();

    /* Calculate the estimated loss rate. */
    double  th_Q_k = bearer_usr_x->GetQueueSize() + 1;
    double  epsilon = 0.1;  /* TODO */
    double  eOverdue_Q_k = eh_VQ_k - el_VQ_k;
    double  overdue_Q_k = h_Q_k - l_Q_k;

    /*
     * Estimated loss rate for incoming packets in queue k and in its virtual
     * queue
     */
    int     eL_Q_k = 0;
    int     N_Q_k = mac_Q_usr_x->GetNbOfTxPacketsBeforeTimeObservation();

    if ((eOverdue_Q_k >= 0) && (eOverdue_Q_k < th_Q_k)) {
        eL_Q_k =
            ((overdue_Q_k + 1) + (eOverdue_Q_k + 1)) /
                (N_Q_k + h_Q_k + el_VQ_k);
    } else if (eOverdue_Q_k >= th_Q_k) {
        eL_Q_k = T_Q_k + epsilon;
    } else {
        eL_Q_k = (overdue_Q_k + 1) / (N_Q_k + h_Q_k);
    }

#ifdef SCHEDULER_DEBUG_PPM
    std::cout <<
        ", l_Q_k=" << l_Q_k <<
        ", h_Q_k=" << h_Q_k <<
        ", N_Q_k=" << N_Q_k <<
        ", eL_Q_k=" << eL_Q_k <<
        ", T_Q_k=" << T_Q_k;
#endif

    if (l_Q_k == h_Q_k) {
#ifdef SCHEDULER_DEBUG_PPM
        std::cout << " --> all in time (Scenario B -> C)" << std::endl;
#endif
    } else if (eL_Q_k >= T_Q_k) {
        /*
         * QoS for packets in queue k cannot be met.
         * Call the cut-in process.
         */
#ifdef SCHEDULER_DEBUG_PPM
        std::cout << " --> cut-in (Scenario B)" << std::endl;
#endif
        AllocCutInUserZForPrbY(usr_x, PRB_y);
    } else {
        /*
         * Discard the overdue packets in queue k.
         * Do not call the cut-in process.
         */
#ifdef SCHEDULER_DEBUG_PPM
        std::cout << " --> discard overdue (Scenario B)" << std::endl;
#endif
        mac_Q_usr_x->DiscardPackets(l_Q_k, h_Q_k);
    }

    SCHEDULER_CHECKPOINT();
}


/*
 ******************************************************************************
 * Phase 3: Cut-in process
 ******************************************************************************
 */
void
DL_PPM_PacketScheduler::AllocCutInUserZForPrbY (int usr_x, int PRB_y)
{
    FlowsToSchedule    *flows = GetFlowsToSchedule();
    int                 num_usrs = flows->size();
    FlowToSchedule     *flow_usr_x = flows->at(usr_x);
    RadioBearer        *bearer_usr_x = flow_usr_x->GetBearer();

    /*
     * Expected throughput if PRB y is located to user x, 0 <= x < n_u, where
     * n_u is the number of users available, 0 <= y <= n_b, where n_b is the
     * number of resource blocks available
     */
    double                                    ET_usr_x_PRB_y   =   bearer_usr_x-
>GetAverageTransmissionRate();
    double              ET_usr_z_PRB_y = 0;
    double              decrease_ET_usr_z_PRB_y = 0;
    double              min_decrease_ET_usr_z_PRB_y = 0;
    int                 C_usr_z_PRB_y = -1;
    bool                first_time = true;

    for (int usr_z = 0; usr_z < num_usrs; usr_z++) {
        /* User z must not be the same as user x */
        if (usr_z != usr_x) {
            ET_usr_z_PRB_y =
                flows->at(usr_z)->GetBearer()->GetAverageTransmissionRate();

            decrease_ET_usr_z_PRB_y = ET_usr_x_PRB_y - ET_usr_z_PRB_y;
            if (first_time == true) {
                min_decrease_ET_usr_z_PRB_y = decrease_ET_usr_z_PRB_y;
                C_usr_z_PRB_y = usr_z;
                first_time = false;
            }
            if (decrease_ET_usr_z_PRB_y < min_decrease_ET_usr_z_PRB_y) {
                min_decrease_ET_usr_z_PRB_y = decrease_ET_usr_z_PRB_y;
                C_usr_z_PRB_y = usr_z;
            }
        }
    }

#ifdef SCHEDULER_DEBUG_PPM
    std::cout <<
        "\tPhase 3: usr_x=" << usr_x <<
        ", PRB_y=" << PRB_y <<
        ", num_usrs=" << num_usrs <<
        ", C_usr_z_PRB_y=" << C_usr_z_PRB_y <<
        ", min_decrease_ET_usr_z_PRB_y=" << min_decrease_ET_usr_z_PRB_y <<
        ", ApplicationID=" <<
        flows->at(C_usr_z_PRB_y)->GetBearer()->GetApplication()-
>GetApplicationID() <<
        std::endl;
#endif

    if (C_usr_z_PRB_y >= 0) {
        /* The y-th RB has been allocated to this user z */
        flows->at(C_usr_z_PRB_y)->GetListOfAllocatedRBs()->push_back(PRB_y);
    }

    SCHEDULER_CHECKPOINT();
}
```

dl-ppm-packet-scheduler.h

```
/*
 *----------------------------------------------------------------------------
 *
 * dl-ppm-packet-scheduler.h
 *
 * The LTE-Sim downlink packet scheduler that implements the Packet Prediction
 * Mechanism (PPM) downlink scheduling algorithm proposed in this paper:
 *
 * "QoS-aware Downlink Packet Scheduling for LTE Networks,"
 * Computer Networks: The International Journal of Computer and
 * Telecommunications Networking archive, Volume 57 Issue 7, May, 2013,
 * Pages 1689-1698
 *
 * Howard Chang
 * howardc@sfu.ca
 * Simon Fraser University
 * 20007-2192
 *
 *----------------------------------------------------------------------------
 */


#ifndef DL_PPM_PACKET_SCHEDULER_H_
#define DL_PPM_PACKET_SCHEDULER_H_

#include "downlink-packet-scheduler.h"


class DL_PPM_PacketScheduler : public DownlinkPacketScheduler {
public:
    DL_PPM_PacketScheduler();
    virtual ~DL_PPM_PacketScheduler();

    virtual void DoSchedule(void);

    void ComputeAverageOfHOLDelays(void);

    virtual double
    ComputeSchedulingMetric(
        RadioBearer    *bearer,
        double          spectralEfficiency,
        int             subChannel);

              void RBsAllocation();

    /* Phase 1 */
    void AllocUserXWithMaxCqiToPrbY(int PRB_y);

    /* Phase 2 */
    void ManageQueuesAndPredictPktDelays(int PRB_y);
    void
    HandleScenarioA(
        int     usr_x,
        int     PRB_y,
        double  dl_bandwidth,
        int    &l_Q_k,
        int    &h_Q_k);
    void
    HandleScenarioB(
        int     usr_x,
        int     PRB_y,
        double  dl_bandwidth,
        int     l_Q_k,
        int     h_Q_k);

    /* Phase 3 */
    void AllocCutInUserZForPrbY(int usr_x, int PRB_y);
};

#endif /* DL_PPM_PACKET_SCHEDULER_H_ */
```