HELSINKI UNIVERSITY OF TECHNOLOGY
Faculty of Electronics, Communications and Automation
Department of Communications and Networking

Yifeng Tan

Active Queue Management for LTE uplink in eNodeB

Master's thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology
Espoo, 5th February, 2009

Supervisor: Professor Riku Jäntti
Instructor: Dr. Sc. Riikka Susitaival

Author: Yifeng Tan

Name of the Thesis: Active Queue Management for LTE uplink in eNodeB

Date: 12th of February 2009 **Number of pages:** 83

Department: Department of Communications and Networking

Professorship: S-72 Communication Engineering

Supervisor: Professor Riku Jäntti

Instructor: Dr. Sc. Riikka Susitaival

Long-Term Evolution (LTE) is an evolved radio access technology of the 3rd generation mobile communication. It provides high peak bit rates and good end-to-end Quality of Service (QoS). Nevertheless, the wireless link is still likely to be the bottleneck of an end-to-end connection. Thus, having a sophisticated method to manage the queues of the mobile terminal is important.

For Wideband Code Division Multiple Access (WCDMA), an Active Queue Management (AQM) algorithm managing the buffer based on the queue size was proposed. In LTE, due to its largely varying bit rates, the queue-size-based approaches are not suitable anymore. Thus, earlier studies have proposed a delay-based AQM to provide a better performance in LTE. For LTE uplink, the existing algorithm is supposed to be implemented in the User Equipment (UE). On the other hand, the implementation of an AQM in the UE is not mandatory. Until now, only a quite simple delay-based queue management method called Packet Data Convergence Protocol (PDCP) discard is standardized by 3GPP. However, this method is not adaptive and cannot thus guarantee a good throughput.

The purpose of this thesis is to develop an AQM method for LTE uplink to enhance the performance of TCP traffic. In order to have a better control of the LTE uplink traffic from the network side, the AQM algorithm is proposed to be implemented in the eNodeB. It retains the delay-based approach; to achieve it, a method is developed to estimate the queuing delays of the UE from the eNodeB side. The delay estimation is based on the changes in Buffer Status Reports (BSRs) and the amount of data delivered in the eNodeB. In LTE, BSRs are created and transmitted by the UE to report the queue length waiting for uplink transmission.

A number of simulations are done to study the performance of the delay estimation and the resulting AQM algorithm. The new AQM algorithm is also compared with other algorithms, i.e., delay-based AQM implemented in the UE, PDCP discard and drop-from-front. The results show that the delay-based algorithm implemented in the eNodeB performs almost as well as when implemented in the UE. The results also show that the advantaged of delay-based algorithms comparing to the drop-from-front and PDCP discard are evident; They maintain a high throughput and the low end-to-end delay in most of the scenarios.

Keywords: AQM, LTE, TCP

Preface

This thesis was done in NomadicLab, which is a part of Ericsson Research.

I would thank my manager Johan Torsner for giving me the chance to do the thesis in NomadicLab. Special thanks to my instructor Dr. Riikka Susitaival who guide me through the whole process; her talent, patience and kindness give me a lot of help. I thank my supervisor Professor Riku Jäntti for supervising the thesis.

I would also thank my parents, my friends and my colleagues, who support me during this time.

Kirkkonummi, 12th of February 2009

Yifeng Tan

Table of content

Abbreviations

3GPP	3rd Generation Partnership Project
ACK	Acknowledgement
AM	Acknowledged Mode
AQM	Active Queue Management
ARP	Address Resolution Protocol
ARQ	Automatic Repeat reQuest
BSC	Base Station Controller
BTS	Base Transceiver Station
CDMA	Code Division Multiple Access
CQI	Channel Quality Indicator
CRC	Cyclic Redundancy Check
cwnd	Congestion window
E-UTRAN	Evolved UTRAN
EDGE	Enhanced Data Rates for GSM Evolution
eNodeB	E-UTRAN NodeB
EPC	Evolved Packet Core
EPS	Evolved Packet System
EUL	Enhanced Uplink
FDD	Frequency Division Duplex
FEC	Forward Error Correction
FTP	File Transfer Protocol

GERAN	GSM EDGE Radio Access Network
GSM	Global System for Mobile Communication
HARQ	Hybrid ARQ
HSDPA	High Speed Donwlink Packet Access
HSPA	High Speed Packet Access
HSPA+	Evolved HSPA
ICMP	Internet Control Message Protocol
IGMP	Internet Group Management Protocol
IP	Internet Protocol
ITU	International Telecommunication Union
LTE	Long-Term Evolution
MAC	Medium Access Control
MIMO	Multiple Input Multiple Output
MME	Mobility Management Entity
MSS	Maximum Segment Size
NAS	Non-Access Stratum
OFDM	Orthogonal Frequency Division Multiplexing
OSI	Open System Interconnection
OSPF	Open Shortest Path First
PAPR	Peak-to-Average Power Ratio
PC	Pipe Capacity
PDCP	Packet Data Convergence Protocol
PDCCH	Physical Downlink Control Channel

PDN-GW	Packet Data Network Gateway
PDPC	Packet Discard Prevention Counter
PDU	Protocol Data Unit
PHY	Physical layer
QoS	Quality of Service
R-AQM	Receiver-AQM
RARP	Reverse Address Resolution Protocol
RAN	Radio Access Network
RED	Random Early Detection
REM	Random Exponential Marking
RLC	Radio Link Control
RNC	Radio Network Controller
ROHC	Robust Header Compression
RRC	Radio Resource Control
RTO	Retransmission Timeout
RTT	Round-Trip Time
S1	The interface between eNodeB and Access Gateway
SACK	Selective ACK
SAE	System Architecture Evolution
SC-FDMA	Single Carrier - Frequency Division Multiple Access
SDU	Service Data Units
S-GW	Serving Gateway
SMTP	Simple Mail Transfer Protocol

T-AQM	Transmitter-AQM
TCP	Transmission Control Protocol
TDD	Time Division Duplex
TM	Transparent Mode
UE	User Equipment
UDP	User Datagram Protocol
UM	Unacknowledged Mode
UMTS	Universal Mobile Telecommunication System
UTRA	Universal Terrestrial Radio Access
UTRAN	Universal Terrestrial Radio Access Network
VoIP	Voice over IP
WCDMA	Wideband CDMA
X2	The interface between eNodeBs

List of Figures

List of Tables

1. Introduction

In the past few decades, mobile communication systems have evolved from 1st generation, which are analog cellular systems, through 2nd generation—known as digital narrowband, to 3rd generation, with higher bandwidth radio interface. Currently, the evolution of 3G is still in progress. The latest commercialized wireless network technologies, called High Speed Packet Access (HSPA) and evolved HSPA (HSPA+), start to be deployed around the world. At the same time, the work of standardization of Long Term Evolution (LTE), started from 2004, is close to ready. LTE provides even higher data rate comparing to Wideband Code Division Multiple Access (WCDMA) and HSPA, with the peak of more than 300 megabits per second in downlink and 50 megabits per second in uplink. Besides, it provides lower user costs, better spectrum efficiency, smaller latency, etc.

Though LTE offers high peak bit rates, the wireless link is still likely to be the bottleneck of an end-to-end connection. In an overload situation, meaning a situation where the incoming data rate to the link is larger than the outgoing data rate from the link, the excessive data is temporarily stored in the memory. If the overload continues, the data queue will accumulate and finally exceed the physical limitation of buffer. Then some data have to be discarded. The straightforward way to handle this problem is discarding the incoming data when the buffer is full. This approach is intuitive and easy to implement, however, it may cause a number of problems e.g. large packets delay, unfair sharing, buffer overflow etc. [3] [20] [21]. In order to overcome these problems, some more sophisticated methods are developed; for example, Active Queue Management (AQM). It drops packets before the buffer is full and thus maintains the queue size and queuing time in an appropriate value.

Considerable work has been done about AQM. E.g., Random Early Detection (RED), proposed by S. Floyd and V. Jacobson in [3], is supposed to be implemented in gateways. Following RED, many other AQM algorithms are proposed, e.g. BLUE is proposed in [5], and Random Exponential Marking (REM) is proposed in [14]. Nevertheless, most of these algorithms are designed for wired networks and are not very suitable for mobile communication due to its varying bandwidth characteristics resulting from varying radio conditions. Packet Discard Prevention Counter (PDPC) algorithm, on the other hand, is an AQM algorithm developed by M. Sågfors et al. and implemented in WCDMA [20] [21]. T. Rathonyi and M. Nilsson also studied one AQM algorithm for HSPA and LTE in [4].

All the AQM algorithms mentioned above are supposed to control the buffer directly at the transmitter, meaning that they should be implemented in User Equipment (UE) side for uplink transmission. On the other hand, the implementation of AQM in the UE is not mandatory. Until now, only a quite simple delay-based queue management method called PDCP discard is standardized by 3GPP [23]. This brings up a potential problem: The UEs with AQM may obtain improved performance, but those UEs without implementing any AQM may still have bad performance. Thus, it is desirable that an AQM algorithm implemented in the eNodeB side can control the queue length of all the UEs.

The purpose of this thesis is to develop an AQM algorithm for LTE uplink in the eNodeB. In this thesis, a “Remote” AQM algorithm is developed. Remote AQM controls the uplink transmission buffer of the UE remotely from the eNodeB side. More specifically, by implementing Remote AQM, the eNodeB maintains the queue size in a reasonable value by dropping received packets when necessary.

The target is to reduce the end-to-end delays and probabilities for buffer overflow and underflow. To achieve this target, a method to estimate the queuing delay of received packets is developed to support the Remote AQM. By simulations, the performance of Remote AQM will be compared to some other queue management algorithms i.e. transmitter side AQM proposed in [4], PDCP discard proposed in [23] and traditional drop-from-front approach.

The rest of the thesis is organized as follows: In Chapter 2-4, background knowledge about LTE, TCP and AQM is introduced, respectively. Chapter 5 introduces the new AQM method and describes it in all details. The simulator and the simulation configuration are briefly introduced in Chapter 6. Chapter 7 presents a detailed result analysis. Some conclusions are drawn in Chapter 8.

2. LTE overview

This chapter, together with the following Chapter 3 and Chapter 4, introduces the theoretical and background knowledge related to this thesis. As mentioned in Chapter 1, LTE is an evolved radio access technology of 3G mobile communication. This chapter will present the overview of 3G evolution and LTE.

2.1. 3G evolution: from WCDMA to LTE

3G is the 3rd generation of standards and technologies for mobile communication. It is based on the standards of International Mobile Telecommunications-2000 (IMT-2000) specified by the International Telecommunication Union (ITU). There are several different kinds of 3G technologies. In Europe, WCDMA was chosen as underlying air interface for Universal Mobile Telecommunications System (UMTS). It was standardized by 3rd Generation Partnership Project (3GPP), which is a collaborative group consisting of international standards organizations and mobile-technology corporations and is still evolving. The WCDMA is specified in 3GPP technical specifications Release 99. This release promises 2 Mbps peak data rate for downlink transmission and 384 kbps for uplink. The next phase of upgrade from WCDMA is High-Speed Downlink Packet Access (HSDPA), which offers up to 14.4 Mbps for downlink. For the uplink, the WCDMA is followed by Enhanced Uplink (EUL) 3GPP Release 6, which provides up to 5.76 Mbps peak data rate. The combination of HSDPA and EUL is referred as HSPA.

The evolution of 3G after HSPA differs from previous Releases: There are two parallel tracks; one is evolved HSPA, also known as HSPA+, which is still an evolution of WCDMA 'with basic WCDMA structures and with a requirement on backwards compatibility to already deployed networks' [1]. The other track is a new radio access technique, known as Long-Term Evolution (LTE), which is based on Orthogonal Frequency Division Multiplex (OFDM). The 3G evolution is presented in Figure 2.1.

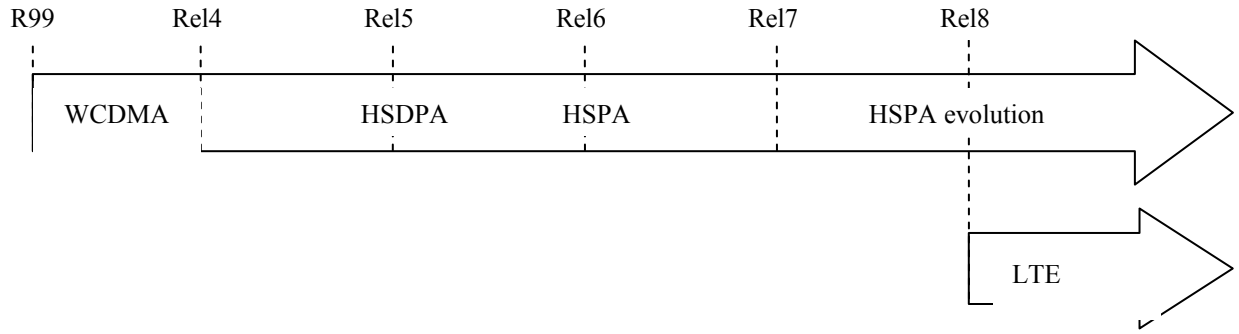

***Figure 2.1**: 3G evolution ([1])*

In parallel to the evolved radio access technology LTE, 3GPP also specifies a new core network evolution, which is called System Architecture evolution (SAE). The purpose of LTE and SAE is to

improve the UMTS system for the future. Standardization of LTE has started November, 2004. The target of LTE/SAE is to provide mobile networks with higher date rate, spectrum efficiency, lower latency, etc. In order to fulfill these requirements, some new techniques are introduced in LTE. For example, Orthogonal Frequency Division Multiplexing (OFDM) is used in downlink data transmission due to its high spectral efficiency and robustness against interference etc. [1]. In uplink, Single Carrier Frequency Division Multiple Access (SC-FDMA) is used because of its lower Peak-to-Average Power Ratio (PAPR) comparing to traditional OFDM [27]. Furthermore, in order to simplify the system architecture and reduce latencies, the LTE radio access network consists only of a single node, called eNodeB.

2.2. LTE design requirements

The requirements of LTE are documented in [26]. They are divided into 7 areas. Main requirement areas and the requirements within the areas are:

1. Capabilities-related requirements

    - The instantaneous downlink peak data rate should be at least 100 Mbps within 20 MHz spectrum allocation and with two receiver antennas at the UE.

    - The instantaneous uplink peak data rate should be at least 50 Mbps within 20 MHz spectrum allocation and with one transmit antenna at the UE.

    - The control-plane latency requirement has two measures, one is the transition time from a camped-state to active state, where the requirement is 100 ms; the other measure is the transition time from a dormant state to active state, where the requirement is 50 ms.

    - The user-plane latency, which is expressed as the time to transmit a small IP packet either from the terminal to the RAN edge node or from the RAN edge node to the terminal measured on the IP layer, is required to be under 5 ms.

2. System performance requirements

    - The average user throughput should be 3 to 4 times more than in Release 6 for downlink and 2 to 3 times more in uplink.

    - The cell-edge user throughput should be 2 to 3 times more than in Release 6 for downlink and 2 to 3 times more in uplink

    - Spectrum efficiency should be 3 to 4 times more than in Release 6 for downlink and 2 to 3 times more in uplink.

3. Deployment-related requirements

    - LTE system can be deployed both standalone and integrating with an existing WCDMA/HSPA and/or Global System for Mobile communications (GSM) network.

- LTE-based radio access can be deployed in both paired and unpaired spectrum allocation. Thus, LTE should support both Frequency Division Duplex (FDD) and Time Division Duplex (TDD) [1].

4. Requirements for architecture and migration to LTE

    - LTE architecture should simplify and minimize the introduced number of interfaces; thus, a single E-UTRAN architecture is required.

    - LTE should be designed to minimize the delay variation for e.g. TCP/IP packet communication.

5. Radio Resource Management requirements

    - LTE should provide Enhanced support for end-to-end Quality of Service (QoS). This requires improved supporting for various applications, services and protocols etc.

    - LTE should provide mechanisms to support operation and transmission of higher layer protocols over radio interface e.g. IP header compression.

6. Complexity requirements

    - Basically, the complexity requirements imply that the number of options should be minimized without redundant mandatory.

7. General requirements

    General requirements address the cost and service related aspects:

    - Cost-related requirements imply that the cost of future network deployment should be minimized. Meanwhile, the existing site locations are able to be used as well. Additionally, the UE complexity and power consumption should be minimized.

    - Service-related requirements address that various types of service should be efficiently supported, including currently available services e.g. Web, File Transfer Protocol (FTP), etc., as well as more advanced services e.g. real-time video.

2.3. LTE/SAE system architecture

As mentioned in Section 2.1, the SAE is the evolution of the new core network. Together with the new radio access technology LTE, it comprises a new cellular system, known as Evolved Packet System (EPS).

As any other cellular system, EPS is made up of the core network and the radio access network, which are called Evolved Packet Core (EPC) and E-UTRAN, respectively. The SAE is illustrated in Figure 2.2.

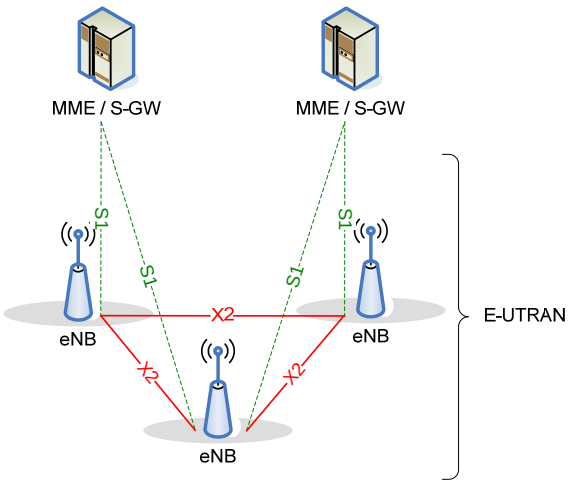

Figure 2.2: *Overall Architecture of EPS ([12])*

EPC, which is composed of several functional entities, e.g. Mobile Management Entity (MME), Serving Gateway (S-GW), Packet Data Network Gateway (PDN-GW) etc., is connected to the E-UTRAN by means of S1 interface. The E-UTRAN, on the other hand, consists of eNodeBs. Among eNodeBs there is an X2 interface interconnecting them to each others. It should be noticed that the S1 and X2 interfaces support many-to-many relation between EPC and eNodeBs, which means one eNodeB can connect to multiple MME/S-GW and multiple eNodeBs at the same time.

EPC provides an access to external data networks and control functionalities, e.g. security, mobility management, charging etc; while the E-UTRAN performs interaction between the mobile terminals, also knows as a User Equipment (UE), and the EPC. For the UE, the functionalities of E-UTRAN can be divided into user plane and control plane protocols, as illustrated in Figure 2.3 and Figure 2.4 below.

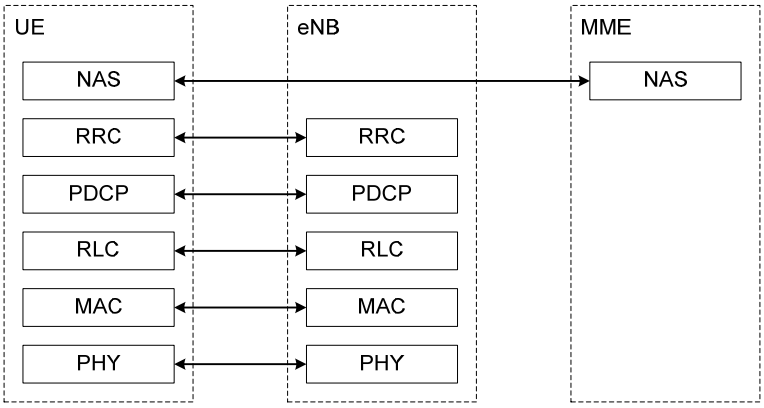

Figure 2.3*: LTE control plane ([12])*

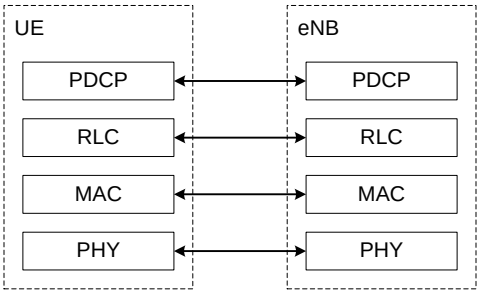

***Figure 2.4**: LTE user plane ([12])*

As Figure 2.3 and Figure 2.4 illustrated, both user plane and control plane consist of layered protocols. For control plane, the protocol stack includes Non-Access Stratum (NAS), which located in UE and MME, and Radio Resource Control (RRC), Packet Data Convergence Protocol (PDCP), Radio Link Control (RLC), Medium Access Control (MAC) and Physical layer (PHY), all of which are located in UE and eNodeB. On the other hand, the user plane only consists of PDCP, RLC, MAC and PHY sub-layers.

The Non-Access Stratum (NAS) layer in MME performs Bearer control, mobility handing for the UE in an idle mode, paging origination as well as security-related functionalities. More specified introduction about the other layers will be presented in the following Section 2.4. The dataflow of the LTE radio interface will be presented in Section 2.6.

2.4. LTE protocol structures

As discussed in Section 2.3, E-UTRAN performs radio access between the core network and the UE. Its functionalities can be structured as different protocol layers. The Figure 2.5 illustrates the protocol architecture for the downlink. The uplink, on the other hand, is very similar to the downlink, except some minor differences, e.g. transport-format selection.

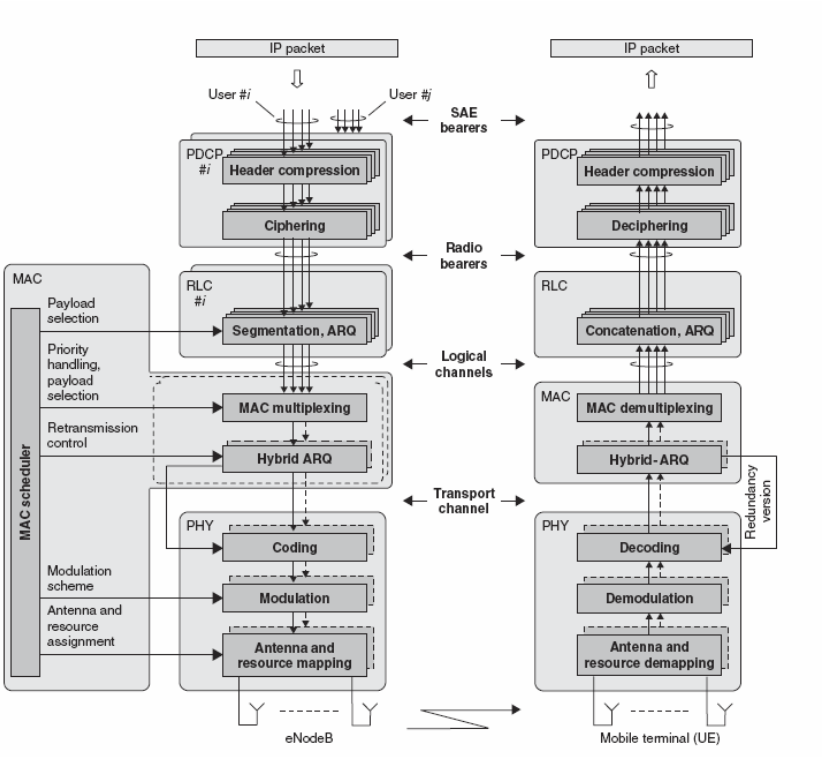

***Figure 2.5**: LTE protocol architecture ([1])*

As can be seen from Figure 2.5, incoming IP packets from the core network go through different layers processed by different functions. The layers and their main functionalities are presented more detailed below.

2.4.1. PDCP

The main functionality of Packet Data Convergence Protocol (PDCP) is IP header compression with the Robust Header Compression (ROHC) mechanism. The purpose of ROHC algorithm is to compress TCP/IP headers of the IP packets and thus reduce the amount of data to be transmitted over the radio interface. For small data units, compressing 40 Bytes header to 3 Bytes results significant savings in the header sizes. Ciphering is another important responsibility of PDCP.

2.4.2. RLC

Radio Link Control (RLC) performs multiple functionalities such as segmentation, concatenation retransmission and in-sequence delivery of data units. The data units coming from PDCP layer, called RLC Service Data Units (SDUs), are segmented or concatenated and then the RLC header is added to them. The resulting data units, RLC Protocol Data Units (PDUs), are then sent to the Medium Access Control (MAC) layer.

RLC can be configured into different modes, depending on what kind of service it is supposed to offer. For those applications for which an error-free transmission is required, e.g., Web and FTP service, RLC should be configured into Acknowledged Mode (AM), in which the Automatic Repeat reQuest (ARQ)

mechanism is used to guarantee the correctness of transmission. On the other hand, when services, for which error-free delivery is less important than the short delivery time, are offered, RLC can be configured into Unacknowledged Mode (UM), in which only the in-sequence delivery is provided but not retransmission are done. Examples of these services are User Datagram Protocol (UDP) traffic or Voice-over-IP (VoIP). The third possible configurable mode is Transparent Mode (TM), in which the RLC is completely transparent and no functions will be performed.

2.4.3. MAC

MAC is responsible for scheduling, hybrid-ARQ (HARQ) retransmissions and logical-channel multiplexing.

- The scheduler is an important part of the MAC layer. It controls resource assignments for both downlink and uplink. In LTE, the downlink and uplink scheduling are independent of each other. Downlink assignments carried on Physical Data Control Channel (PDCCH) determines which UEs are supposed to receive which transmitted blocks. Besides, the downlink scheduler also decides modulation scheme, transport-block size, etc, which means that it also affects the RLC and physical layers, as illustrated in Figure 2.5. The uplink scheduler, on the other hand, is somewhat similar to the downlink scheduler, but for the reverse direction, which is from the UEs to the eNodeB. An important entity that should be noticed here is the Buffer Status Report (BSR), which is created and transmitted by a UE. BSRs report the amount of data waiting to be transmitted for uplink transmission. Some more detailed discussion on scheduling and BSR will be presented in Section 2.5.

- HARQ with soft combining is used to provide robustness against transmission errors. In this mechanism, multiple parallel stop-and-wait processes are progressed at the same time, combining Forward Error Correction (FEC) with standard ARQ. When the received data cannot be correctly decoded, the receiver asks a retransmission. Then error correction at the receiver can attempt to combine the transport blocks received from both transmissions. This mechanism guarantees both good throughput and reliability of transmission [1].

- As seen from Figure 2.5, the MAC layer serves RLC in the form of logical channels. On the other hand, the MAC layer is served by PHY layer in the form of transport channels. Thus, an important functionality of MAC layer is to multiplex different logical channels and map the transport channels to appropriate logical channels.

2.4.4. PHY

The physical layer offers data transport service to upper layers. More specifically, it is responsible for channel coding/decoding, modulation/demodulation, physical-layer HARQ, frequency and time synchronization, multi-antenna processing, mapping of the coded transport channel onto physical channels, etc. The physical layer model of LTE is illustrated in Figure 2.6 below.

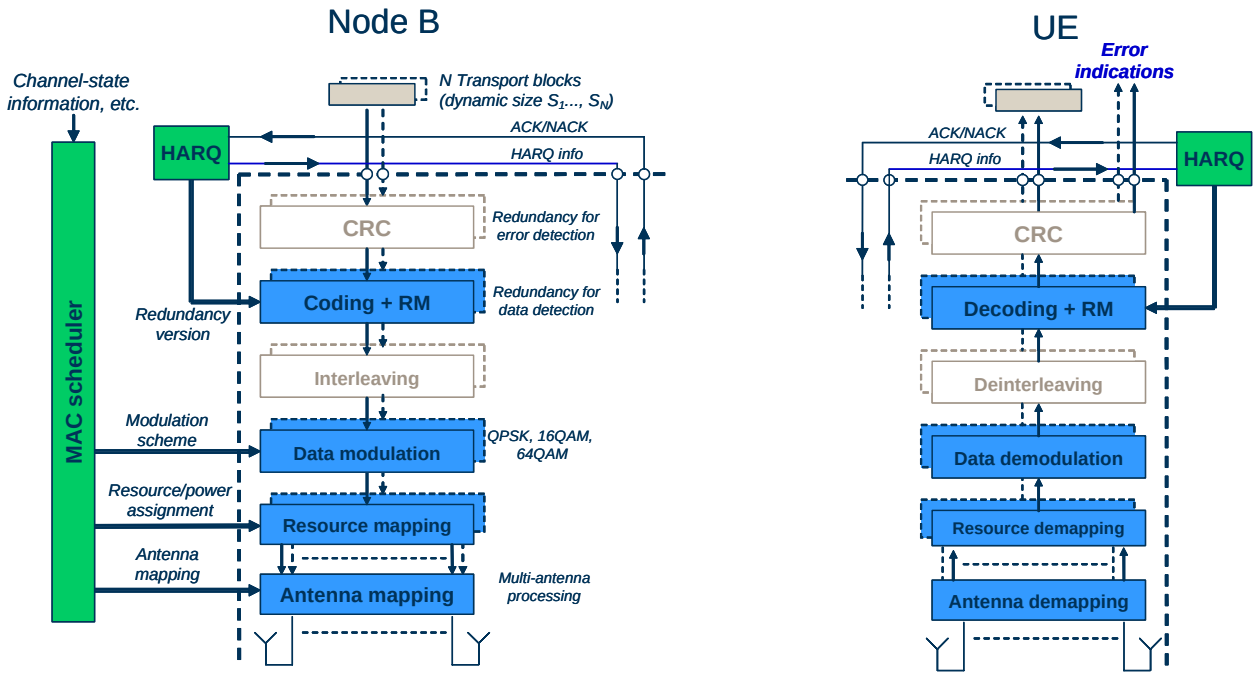

***Figure 2.6**: Physical-layer model for DL-SCH transmission ([11])*

Unlike in previous 3GPP releases, e.g. WCDMA, HSPA, in which downlink transmission is based on CDMA technology, the LTE downlink scheme is based on another multiplexing technique called OFDM. OFDM is a kind of multi-carrier technology but with a large number of sub-carriers which are, at least in one cell, mutually orthogonal. By spreading the data on a large number of sub-carriers, a wideband channel is split into many narrow band channels. By this way, the "selective fading" problem common in wideband mobile communication can be solved to some extent [1]. On the other hand, keeping the sub-carriers orthogonal can theoretically eliminate the interference between sub-carriers thus the guard band between sub-carriers are not required; as a consequent, the orthogonality of sub-carriers can sufficiently reduce the waste of bandwidth [27].

The basic OFDM time-frequency grid of LTE downlink is illustrated in Figure 2.7. As the figure shows, the sub-carrier spacing Δf has been chosen to 15 kHz. Furthermore, 12 consecutive sub-carriers comprise one downlink resource block, with the bandwidth of 180 kHz.

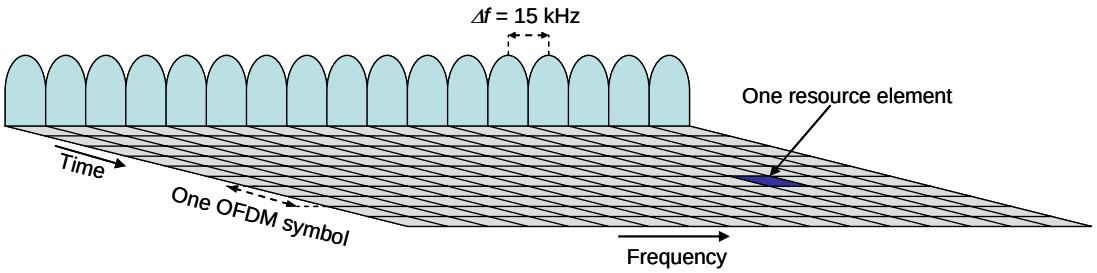

***Figure 2.7**: LTE downlink physical resource ([1])*

Although OFDM works fine in downlink transmission, for LTE uplink, the situation is different. One obvious drawback of OFDM is the large variation of instantaneous transmitting powers, also known as high Peak-Average-Power Ratio (PAPR). Variation significantly decreases the power-efficiency and increases the Bit Error Rate (BER) [27]. On the downlink, the high PAPR problem might be easy to overcome by using special power amplifier and some other sophisticated mechanisms. However, those mechanisms increase the computational complexity and power consumption. But on the uplink, the power control is critical for mobile terminals. Thus, the traditional OFDM is not quite suitable. Instead, Single-Carrier FDMA is selected for LTE uplink due to its lower PAPR [1]. The basic idea of this scheme is spreading of the symbols of the uplink on a group of sub-carriers through the use of a unitary transformation [27]. By this way the peak-to-average ratio of the signals can be considerably reduced, leading to the increased coverage and decreased terminal power consumption. In addition, similar to downlink transmission, the uplink is also based on orthogonal separation of users in time and frequency domain, so the intra-cell interference can be avoided.

2.5. Scheduling

Scheduling determines how the available radio resources in the mobile communication system are shared between the different UEs. In LTE, the basic strategy of scheduling is so-called *dynamic scheduling* [1], where the eNodeB makes a scheduling decision per each TTI and informs it to the selected UEs. As discussed in Section 2.4.3, the downlink and uplink scheduling are independent of each other. This section will discuss both downlink and uplink scheduling respectively.

2.5.1. Downlink scheduling

Downlink scheduling is responsible in controlling for which UE(s) the eNodeB transmits and with which resource blocks. In addition, the downlink scheduling is responsible for the selection of the transport-format e.g. transport-block size, modulation scheme, antenna mapping, etc. and logical channel multiplexing for downlink transmission [1].

One of the key characteristic of mobile communication is the rapid and significant variation of the wireless link environment, both on time domain and frequency domain. Therefore, in order to optimize the scheduling decision, instantaneous channel quality is taken into account; this strategy is called *channel-dependent* scheduling. The overall target of channel-dependent scheduling is to take advantage of the channel condition variation between UEs and preferably schedule resource to UEs with best channel condition. To obtain the information of instantaneous channel quality, Channel Quality Indicator (CQI) is created by the UE and sent to the eNodeB to indicate the channel quality in both time and frequency domain. The CQI is based on the measurement of downlink reference signals. Figure 2.8 illustrates the general procedure of downlink scheduling.

In addition to supply efficient radio resource utilization, scheduling is also responsible for handling priority issues, due to different requirements of QoS. For example, the data from VoIP, which requires small end-to-end packet delay, is supposed to have higher priority than the data from a file transferring application e.g. FTP. To support the priority issue, a set of priority queues can be defined. The data is classified to different queues according to the priority.

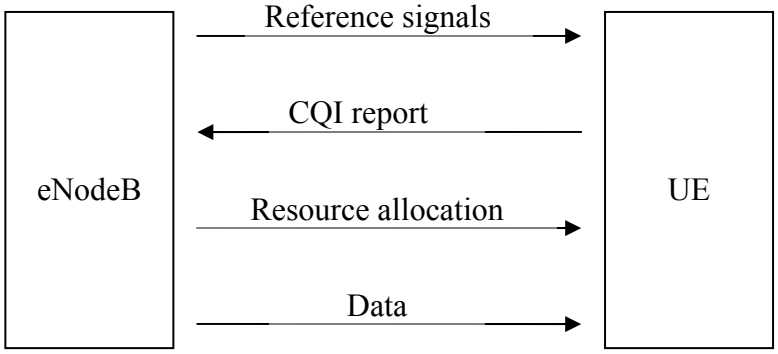

Figure 2.8: *General procedure of downlink scheduling*

2.5.2. Uplink scheduling

Similar to downlink scheduling, uplink scheduling determines which UE(s) is allowed to transmit and with which resource blocks during a given time interval. However, unlike downlink scheduling, uplink scheduling cannot automatically "know" the transmission demand from a UE. Thus, before the UE can transmit data to the eNodeB, it first sends a Scheduling Request (SR) to request the transmission permit. When the scheduler in eNodeB receives the SR, it replies with a scheduling grant for the request. In addition, the scheduler determines the time/frequency resource which the UE should use as well as transport format. After the UE has received the information of assignment, i.e., the UL grant, it can transmit data with required parameters over a sub-frame when the grant is valid. Additionally, an entity called *Buffer Status Report (BSR)* can be transmitted together with the data indicating the queue length of the UE. The general procedure of uplink scheduling is illustrated in Figure 2.9.

According to [23], there are several events to trigger the BSR transmission:

- New data arrives to the UE transmission buffer where the data is with higher priority than the data already existing in the buffer.
- Uplink resource has been allocated and the size of padding bits is larger than the size of BSR. The padding bits refer to the bits in the transport format that are over the actual need.
- The UE moves from one cell to another.
- The Periodic BSR timer, which is used to trigger Periodic BSR, expires.

A triggered BSR can also be cancelled in case when the allocated resource is large enough to accommodate all the data in buffer excluding the size of BSR.

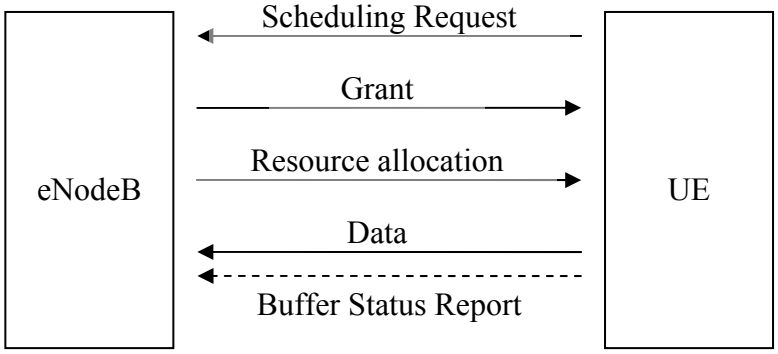

Figure 2.9: *General procedure of uplink scheduling*

2.6. Dataflow

According to the discussion above, an example of downlink data flow of LTE radio interface is illustrated in Figure 2.10 below. The uplink is somewhat similar. In Figure 2.10, three IP packets are considered, two on the radio bearer 1 and one on the radio bearer 2. These IP packets are passed first through the PDCP layer with functionalities of IP head compression and ciphering. Then PDCP header is added to each packet. The PDCP header carries the information required to deciphering the PDU in the UE. The output from the PDCP, which is so-called PDCP PDU, is fed to RLC.

The RLC layer performs segmentation and concatenation of PDCP PDUs and then adds an RLC header. The header is used to identify the RLC PDU when a retransmission is required. In addition, the header can also be used to support in-sequence delivery in the UE. The RLC PDUs are forwarded to MAC layer. The multiplexing functionality performed by MAC can assemble a number of RLC PDUs to form a MAC SDU. The RLC PDUs can be from the same of different radio bearers. After attachment of a MAC header to the MAC SDU, the data unit is now called a transport block. The size of transport block depends on the instantaneous data rate. Finally, the transport block is attached with a Cyclic Redundancy Check (CRC), which is used for error-detection, and transmitted over air.

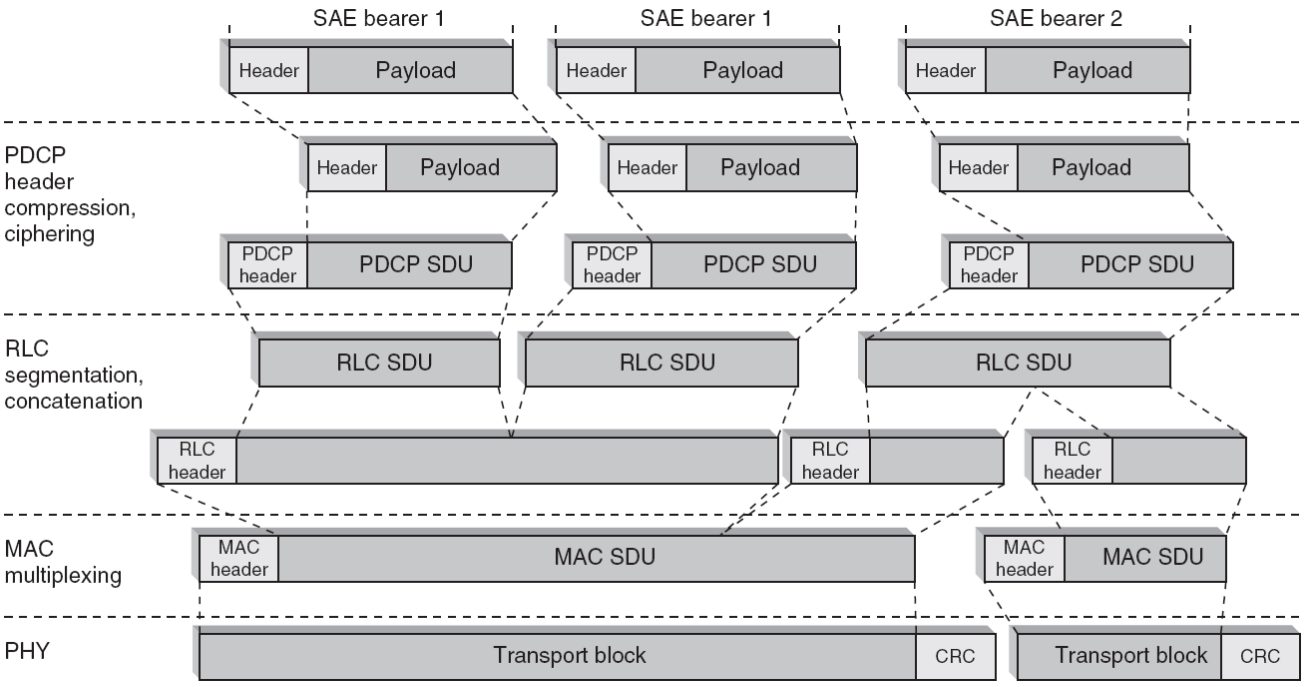

Figure 2.10: *LTE data flow ([1])*

3. Introduction to TCP

As discussed in [1], one major driving force of 3G evolution from WCDMA to LTE is the increasing requirements to access the Internet with high speed by mobile terminals. Meanwhile, the Transmission Control Protocol (TCP) is one of the core protocols of the Internet. In this section, some basic knowledge about TCP is introduced.

3.1. Internet Protocol Suite

Internet Protocol Suite, commonly called TCP/IP, is the most widely used protocol suite that runs on Internet and other commercial networks. It captures 4 layers in the Open Systems Interconnection Reference Model (OSI model), as illustrated in Table 3.1.

Table 3.1: TCP/IP protocol model

Layers	**Functionalities**	**Protocols**
Application Layer	Supporting network applications	Telnet, FTP, SMTP, etc
Transport Layer	Transporting up-layer message between source and destination	TCP, UDP, etc
Internet Layer	Routing from host to host	IP, ICMP, IGMP, etc
Link Layer	Interface with physical media	ARP, RARP, OSPF, etc

In this section only the protocols tightly related to the thesis are discussed, i.e., Internet Protocol (IP) and Transmission Control Protocol (TCP). More detailed information about Internet Protocol Suite can be found from [10].

As presented in [1], EPC is an all-IP system. However, IP only provides unreliable service, which means the packets can be lost, corrupted, duplicated or delivered in disorder. TCP, on the other hand, provides reliable, in-order delivery service on the top of IP. Besides, TCP is also responsible for flow control, congestion control, connection management etc.

3.2. Basic behavior and concept of TCP

TCP is a connection-oriented protocol, which means that a virtual connection will be established before two end hosts can really communicate to each other. TCP uses a mechanism called “handshake” to establish the virtual connection. More specifically, the host asking for the connection, also known as a *client*, sends first one special segment, also known as *server to request connection*, to the host. After the server has received the connection request segment, it responds and sends another special segment to grant or not grant the connection. If the connection is granted, the client sends the third special segment to acknowledge the connection-granted segment and the connection is finally established. This

procedure is so-call three-way handshake. Some common parameters, such as initial sequence number, are also set during the handshake procedure.

The termination of connection is similar to the establishment, except that termination requests are sent through both sides due to the fact that TCP connection is full-duplex. It is also worth to know that either side can send a termination request.

The TCP connection establishment and termination are illustrated in Figure 3.1.

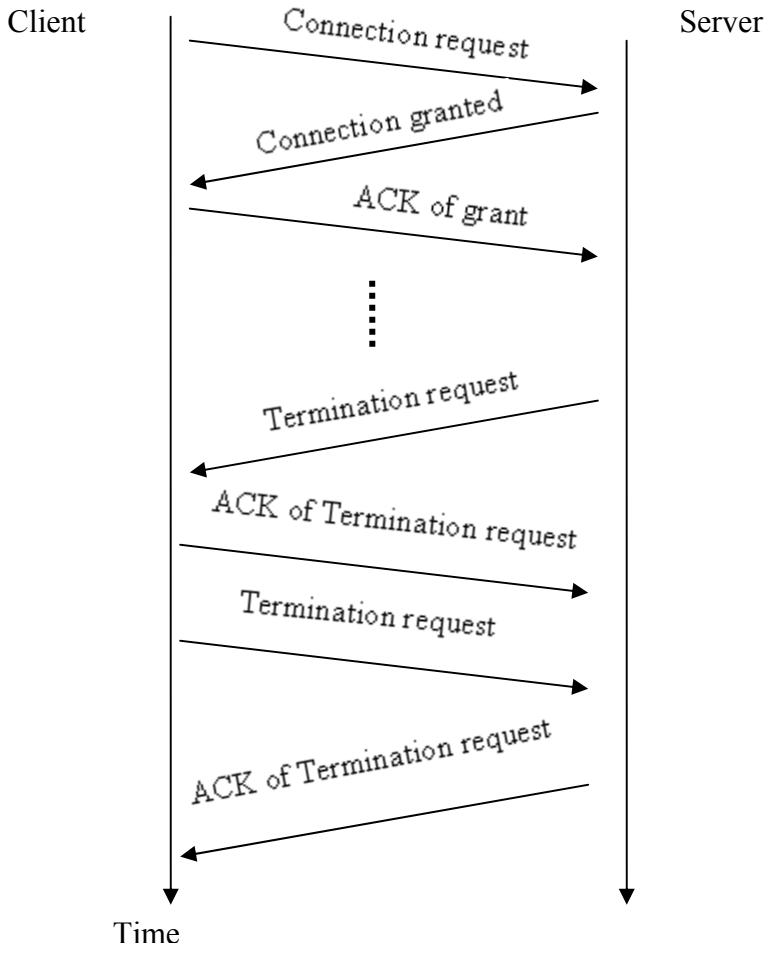

Figure 3.1: *TCP connection establishment and termination ([10])*

As presented in Section 3.1, TCP is designed for reliable data transfer. To achieve this target, it also uses the ARQ mechanism, which has been mentioned in Section 2.4.2. The basic idea of ARQ is that every time when the receiver receives a packet correctly, it sends an *acknowledgement* to the transmitter. If the transmitter does not receive the acknowledgement of some packet before the *timeout*, it retransmits the corresponding packet. This procedure is repeated until all packets are acknowledged. This guarantees the correctness of the transmission.

The TCP algorithm used nowadays is much more complicated than the basic ARQ, but the principal idea is still the same. TCP breaks the data stream from an upper layer into segments, adds a TCP header to each segment and assigns a sequence number as segments "identity". By using ARQ, it is not only guaranteed that every single segment is received correctly but also they are in right sequence. One important improvement of TCP comparing to basic ARQ is the use of pipelining, which means that the sender is allowed to transmit multiple segments that are not acknowledged yet. It greatly increases the throughput of TCP transmission.

There are several important parameters for TCP:

- *MSS* − The largest TCP segment that can be transmitted by the sender.
- *RTT* − The time it takes from transmitting a segment until receiving the acknowledgement. Because the conditions in networks are always changing, the RTT varies from segment to segment. In order to get a typical RTT, not an instantaneous one, the following smooth algorithm is used;

  $$\text{Estimated\_RTT} = (1 - \alpha) * \text{Instantaneous\_RTT} + \alpha * \text{Estimated\_RTT},$$

  where α is a smoothing factor which is recommended to 0.9 [10].

- *RTO* − The time the sender will wait until a retransmission is triggered if the acknowledgement is not received. It is calculated depending on Estimated_RTT.

  $$\text{RTO} = \text{Estimated\_RTT} + 4 * \text{Dev},$$

  where Dev is the mean deviation of Estimated_RTT defined as:

  $$\text{Deviation} = (1 - \beta) * \text{Deviation} + \beta * | \text{Instantaneous\_RTT} - \text{Estimated\_RTT} |,$$

  where β is another smoothing factor recommended to 0.25 [10].

3.3. TCP congestion control

In data networks, the volumes of data streams vary from time to time. At certain time, some links or nodes may have more data than they are capable to deal with. This is so-call network congestion. Occasionally the incoming data rate is greater than that outgoing; therefore the router has to store the data in the memory temporarily waiting for transmission later. If this procedure lasts too long, the memory will eventually run out and some data have to be discarded. There are several options to do when facing this situation. The simplest ones are passive mechanisms e.g. drop-tail or drop-front, which means that the incoming packets (drop-tail) or the longest stayed packets (drop-front) will be discarded. More complicated methods, usually with better performance, are Active Queue management mechanisms, which will be discussed in Chapter 4.

When some packets discarded, the sender will finally be aware of the loss, due to the acknowledging transmission mechanism. For example, if no ACK is received after a while or several duplicated acknowledgements are received, the loss is most probably occurred. The sender then retransmits those

packets, but it also takes some other actions to avoid further congestion, which is known as *TCP congestion control*. There are a few different kinds of congestion control algorithms, which are implemented in different TCP versions.

Before the discussion of congestion control algorithms, two "windows" should be introduced:

- *Congestion window (cwnd)* − As discussed in Section 3.2, one improvement of the TCP is allowing the sender to transmit multiple segments "in the air", i.e. send those segments without any acknowledgement received by itself. The allowed number of non-acknowledged segments is indicated by the congestion window. In other words, the congestion window decides how many segments can be transmitted in the air by the sender.

- *Advertised window* − This window indicates the amount of data the receiver can process at a time, i.e., the advertised window is a receiver side window of how many segments can be transmitted in the air.

As a result, the amount of actual segments that can be allowed to be transmitted in the air is the minimum of the congestion window and the advertised window. However, only the congestion window is considered in this section.

Beside the congestion window, another important concept related to the TCP performance is the pipe capacity (PC), also known as bandwidth-delay product. PC is the minimum load that a data link needs to have in the air in order to fully utilize the available bandwidth. The PC is calculated as

$$PC = Bandwidth * RTT_{e2e}.$$

When the congestion window is smaller than the PC, the link is under-utilized, which will decrease the throughput. Congestion control is also used to avoid this situation.

As discussed above, the purpose of congestion control is to decrease the possibility of network congestion as well as link under-utilization by regulating the congestion window. In the following subsections we will see how the different congestion control algorithms achieve the purpose.

3.3.1. Tahoe

Tahoe is a wildly implemented TCP congestion control algorithm originally proposed by V. Jacobson in [2]. Tahoe implements several important algorithms like slow start, congestion avoidance, known also as "Additive Increase Multiplicative Decrease", and Fast Retransmit:

- Slow start

  The slow start algorithm initially set the *cwnd* to 1 or to a few segments. Every time the TCP sender receives an acknowledgement, it increases the *cwnd* by 1. This behavior doubles the *cwnd* every RTT and let the *cwnd* increase exponentially. This behavior continues until the *cwnd* reaches some threshold or the ACK for some segment is not received, which indicates a

segment loss. For the slow start algorithm, a way to detect segment loss is that the RTO timer expires before the required ACK is received.

When a loss occurs, half of the current *cwnd* is saved as *slow start threshold (ssthresh);* at the same time, the *cwnd* is set to initial value and increases exponentially again. Once the *cwnd* reaches *ssthresh,* TCP enters Congestion Avoidance phase which will be introduced in the following paragraph.

- Congestion Avoidance

  When the TCP enters the congestion avoidance phase, the *cwnd* is increased by *1/ cwnd* always when an ACK is received. As a result, the increase of *cwnd* is additive as compared to slow start's exponential increase.

- Fast Retransmit

  Before the Fast Retransmit is explained, it is worthy to know that the ACK message includes the sequence number of the segment that should be transmitted next. Therefore, if there is a segment loss or out-of-order delivery, the sender will receive duplicate ACKs, which are requesting the same segment. Fast Retransmit algorithm defines another way to indicate segment loss by receiving three duplicate ACKs. Thus, when the TCP sender receives the third duplicate ACK, it retransmits the segment the receiver asks for. At the same time, the sender also save the *ssthresh* and set the *cwnd* to the initial value, as the slow start algorithm does.

As discussed above, for TCP Tahoe, there are two ways to detect the segment loss. One way is to detect that the RTO for the lost segment expires, and the other is to detect that the sender receives three duplicate ACKs. Whenever the sender is aware of a segment drop, it retransmits the lost segment. At the same time, the slow start algorithm is triggered.

One problem of Tahoe is that when the slow start is triggered, the pipeline is usually empty causing the under-utilization of the link.

3.3.2. Reno

Reno, as proposed in [6], retains the basic principles of Tahoe. However, it adds a mechanism called "fast recovery" to prevent from the pipe under-utilization:

- Fast Recovery

  Fast Recovery is an improvement of fast retransmit. When the third duplicate ACK is received and the missing segment is retransmitted, *ssthresh* is saved as half of current *cwnd*, the same as in fast retransmit algorithm; however, the fast recovery does not set the *cwnd* to initial value, instead, the *cwnd* is set to *ssthresh* plus 3. Additionally, every time when another duplicate ACK is received, *cwnd* is incremented by one. If a received ACK is acknowledging a new segment instead of a retransmitted segment, *cwnd* is set to *ssthresh*.

According to [19], the Performance of Reno is better than that of Tahoe. The major factor is that when the third duplicate ACK is received, Tahoe set *cwnd* to initial value, which is 1 or a few segments. This causes the pipe under-utilization. Meanwhile, Reno sets *cwnd* to *ssthresh* plus 3, which improves the throughput considerably. However, Reno has severe problems when multiple segments are dropped at the same window [19].

3.3.3. SACK

Both versions of TCP discussed above, Tahoe and Reno, use cumulative acknowledgements, which means that only limited amount of information is available for the TCP sender and, in one RTT, only one packet loss can be learned by the sender. So when multiple segments are lost in a window, the TCP sender will most likely to get a retransmission timeout and the slow start is trigged, which leads to throughput reduction. In order to overcome this problem, Selective Acknowledgement (SACK) is proposed in [13]. The most important improvement of SACK is that it can acknowledge blocks of isolated segments which have been successfully received. This decreases the possibility that a slow-start is triggered when there are multiple segments lost in one window improving the throughput considerably.

3.4. TCP over wireless network

TCP is originally designed for wired networks; however, in wireless networks, the data transmission characteristics are much different. The signal strength can vary a lot because it depends on a number of factors, e.g. distance from the antenna, velocity of terminal etc. For that reason, in most wireless access technologies, automatic bit rate control schemes are implemented to adjust the bit rate of transmission. Still, the TCP is not adaptive enough as regard to remaining rate diversity. When the bit rate changes considerably, problems may arise causing the performance degradation [15] [16][17].

4. Introduction to AQM

Chapter 3 has introduced the basic concepts of TCP. Active Queue Management (AQM), one the other hand, is a mechanism designed to improve the TCP performance. In this chapter, some basic ideas about AQM will be presented. Furthermore, three detailed algorithms will be introduced.

As mentioned in Section 3.3, when the amount of queuing data in the buffer exceeds the physical limit, some packets have to be dropped. The most straightforward method of doing this is drop-tail, which means all the incoming packets are discarded because there is no "room" for them. This method is intuitive and easy to implement, however, it suffers performance degradation in some situations. One possible problem is known as *unfair sharing*. Unfair sharing happens when one or several TCP flows occupy the whole buffer so that new incoming flows are locked out.

To avoid unfair sharing, new methods called drop-from-front and random drop are proposed. The differences of these methods relate to way how the packets to be discarded are picked up when the buffer is full. Instead of dropping incoming packets, drop-from-front drops the packets which have stayed the longest time in the queue, while the random drop picks the packets to be discarded randomly. These methods indeed improve the TCP performance as compared to drop-tail. However, they still suffer from fact that they are passive in responding to congestion, which means that only when the buffer is full, something is done.

Because passive queue management allows queues to be full or almost full, one obviously problematic situation occurs when a burst of packets arrives when the buffer is almost full. Then multiple packets have to be discarded. If those packets belong to different TCP flows, all those flows might reduce the sending rate at the same time or even trigger the slow-start phase. This is known as *global synchronization*. Global synchronization can decrease the throughput significantly; drawbacks are even more severe in radio networks because of the variation of data rates and thus a higher probability to packets dropping,

One straightforward way to avoid the global synchronization is increasing the physical buffer; nevertheless, implementing large buffers do not only increase the system cost but also causes other problems, e.g. significant delay, timeouts inflation, viscount web surfing [20] [21], etc. Thus, more sophisticated mechanisms are needed to control the queue.

Active Queue Management (AQM) can somehow overcome the drawbacks of passive queue management by maintaining the queue size, queuing time or both of them within a reasonable value. Considerable work has been done in this field and several algorithms have been proposed. RED and its variations are the best known algorithms of AQM [3] [22] and BLUE, first proposed in [5], is another important algorithm. All those algorithms are supposed to be implemented in gateways of the Internet, usually meaning a wired network. Nevertheless, as discussed in Section 3.4, wireless networks have different characteristics than wired networks and should be thus studied separately. Fortunately also this work has attracted much attention. PDPC [20] [21] , developed by M. Sågfors et al, is implemented in 3G WCDMA links. Another AQM algorithm, also developed originally in Ericsson, is studied in detail and proposed to be implemented in HSPA by T. Rathonyi and M. Nilsson in [4]. Unlike other

algorithms mentioned above, the algorithm studied in [4] is delay-based. In next subsections the AQM algorithms mentioned here are introduced in more detail.

4.1. RED

The RED algorithm is wildly accepted and implemented AQM method in current Internet routers. It is a queue-size-based algorithm defining two threshold called min_{th} and max_{th}. When the queue size is less than min_{th}, every coming packet is taken into the queue, and when the queue size is greater than min_{th} but less than max_{th}, every incoming packet have a probability p_α to be dropped, where p_α is a value with range from 0 to max_p. If the queue size is greater than max_{th}, then every coming packet will be dropped. In order to avoid unnecessary dropping when packets arrive in a burst, an average queue size *Avg*, is used instead of the current queue size.

The general RED algorithm is give below as Algorithm 4.1.

Algorithm 4.1

for each packet arrival
    calculate the average queue size Avg
    if $min_{th} \le Avg \le max_{th}$
        calculate drop probability p_α
        with probability p_α*:*
            mark (or drop) the arriving packet
    else if $Avg \ge max_{th}$
        mark (or drop) the arriving packet

As seen from the Algorithm 4.1, The RED algorithm actually has two separate procedures: one is computing the average queue size to determine the degree of burst; the other is computing the probability to drop the incoming packet, which depends on the previous calculated average queue size. The *Avg* is calculated using the equation:

$$Avg = (1 - w_q) * previous\_Avg + w_q * current\_size,$$

where w_q is a queue weight having a value between 0 and 1. The smaller the value of w_q, the smoother averaging of *Avg*, meaning that incoming packets can be maintained in a burst without significant increase in the dropping probability.

The p_α, on the other hand, is more complicated to calculate. To calculate p_α, another related parameter p_b is calculated depending on max_p, min_{th} and max_{th}:

$$p_b \leftarrow max_p(avg - min_{th})/(max_{th} - min_{th})$$

Then the drop probability p_α is calculated as

$$p_a \leftarrow p_b / (1 - count \times p_b),$$

where *count* is the number of packet since last drop.

4.2. PDPC

The RED algorithm is certainly an improvement over drop-tail; however, it may be not the best choice for wireless links. As discussed in [4] [20] [21], Wireless links in WCDMA and HSPA have their own characteristics e.g. the link queue is dedicated to only one user, the wireless link is the main contributor of the end-to-end round trip time etc. Thus Ericsson has developed an AQM algorithm called the Packet Discard Prevention Counter (PDPC) to be implemented in WCDMA.

As RED, PDPC also uses the queue size as an indicator of congestion, but it operates with the instantaneous queue size instead of the average queue size used by RED. Another difference is that PDPC discards packets with deterministic drops comparing to the probabilistic policy in RED. In addition, a counter is used to prevent consecutive drops in one TCP window.

There are three parameters in this algorithm, T_{min} , T_{max}, and *C (counter)*. Each time a new packet arrives, the algorithm will decide if this packet is accepted or not, according to the instantaneous queue size and parameters T_{min} , T_{max}, and *C*.. That is, if the queue size is smaller than the T_{min}, the new arriving packet is accepted; if the queue size is larger than T_{min} but smaller than T_{max}, the packet is discarded only if the *C* is 0; and if the queue size is larger than T_{max}, the incoming packet is discarded anyway, regardless of the *C*. The counter *C* is used to prevent discarding multiple packets consecutively. It is initially set to 0. When a packet is discarded, *C* is set to be a predefined number *n*. Then for each incoming packet, *C* decreases by one until it becomes 0 again.

The setting of the parameters is also proposed in [20] [21]. The T_{min} should be set to at least the pipe capacity to avoid the link under-utilization. However, recall that the pipe capacity is the product of RTT_{e2e} and *bandwidth*, and the bandwidth of wireless links various fast. For simplicity, the T_{min} is calculated according to some typical value of bandwidth. Once the T_{min} is calculated, the T_{max} and *C* can be calculated according to T_{min}. In [20] [21], T_{max} is proposed to four times of T_{min} and *C* two time of T_{min}. PDPC works as Figure 4.1 shows.

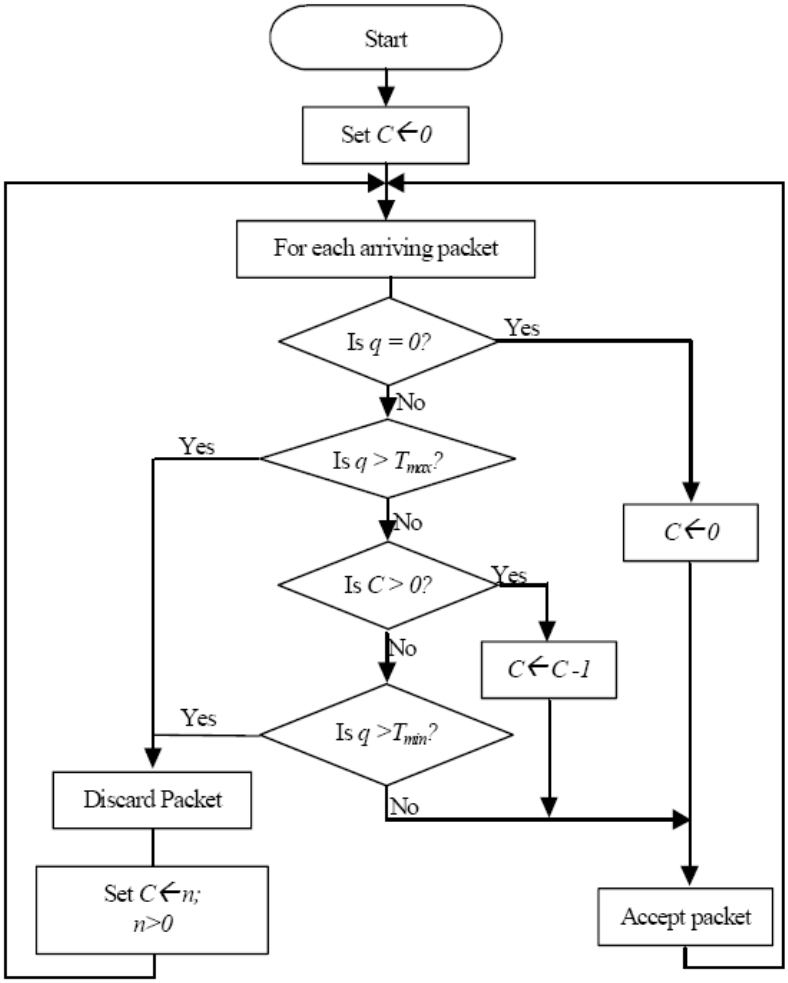

Figure 4.1: *The PDCP algorithm ([20])*

4.3. A delay-based AQM

PDPC worked well in WCDMA, but when the 3G evolved to HSPA with a higher peak data rate, the situation changed. As discussed in [4], a larger bandwidth variation of the latter network significantly degrades the performance of PDPC. Thus, a new AQM algorithm is studied in [4]. The algorithm is delay-based, which means that it decides the packets to be dropped based on the time they have been in the queue.

There are four parameters in this algorithm: *minAgeThreshold*, *maxAgeThreshold*, *lowerDropThreshold* and *minInterDropTime*. As discussed in Section 4.2 and [20], one of the most important purposes of AQM over 3G links is preventing link under-utilization; that is why the T_{min} is used in PDPC algorithm. In delay-based AQM, a parameter called *minAgeThreshold* is used with similar purpose to T_{min}. But unlike T_{min}, which is determined by the pipe capacity of the link, the *minAgeThreshold* is determined by end-to-end Round Trip Time (RTT_{e2e}) of the link. According to [4], the RTT_{e2e} is less dependent on the instantaneous bandwidth than Pipe Capacity. As a consequence, the

minAgeThreshold is less dependent on bandwidth than T_{min}; therefore, the delay-based AQM usually has better throughput in circumstance of large varying bandwidth [4].

The *minInterDropTime* is defined to prevent the dropping multiple packets consecutively, for the same reason as the counter in PDPC. The *lowerDropThreshold* is used to make sure that the algorithm does not drop from an almost drained queue, no matter how long time the packets have been in the buffer. Another reason for this parameter is ensuring that there is sufficient number of packets left in the queue to trigger duplicate ACKs after the drop and thereby avoid RTO, if the data belongs to some TCP flow. The *maxAgeThreshold* defines the algorithm drops the packets that have stayed in the queue longer than the parameter, regardless of the *minInterDropTime*. To this end, the AQM algorithm can be defined as Algorithm 4.2

Algorithm 4.2

```
for each outgoing packet
    if (size ≤ lowerDropThreshold)
        transmit packet
    else
        if ( ( delay > minAgeThreshold AND now − previousDropTime > minInterDropTime ) )
          OR  (delay > maxAgeThreshold ) )
            discard packet
            previousDropTime = now
        else
            transmit packet
```

5. AQM in eNodeB for uplink traffic

In this thesis, a new AQM algorithm for LTE uplink will be developed and investigated. The main target of this algorithm is to improve the performance of TCP flows when the file is uploaded from the terminal to the server located in a fixed network. Chapter 2-4 has introduced related background and theoretical issues. In this chapter, a method to estimate the packet queuing delay of the UE buffer from the eNodeB side is introduced. Served by this method, a new AQM method is proposed at the end of this chapter.

5.1. Introduction

Chapter 4 introduces several AQM algorithms and their typical implementing circumstances. PDPC is an algorithm which discards packets based on the queue length in the buffer. For LTE it is not suitable due to the largely varying bit rates [4]. On the other hand, PDCP discard [23] is a delay-based algorithm which discards packets based on how long time the packets have been in the queue. When the delay exceeds some predefined threshold, the packets will be discarded, as illustrated in Algorithm 5.2.

Algorithm 5.1: PDCP discard

```
for each outgoing packet
    if ( delay > delayThreshold )
        discard packet
    else
        transmit packet
```

This algorithm can maintain small end-to-end delay but may have considerable throughput degradation in some situations, e.g. consecutive packet drops etc. More detailed performance of PDCP discard will be illustrated in Chapter 7. The AQM algorithms proposed in [4], on the other hand, retains the delay-based principle and implements some mechanisms to improve the throughput, e.g. preventing from consecutive packet drops or no drop when queue is small etc., without introducing large end-to-end delay. The simulation results in [4] shows that the algorithm performs well in EUL; thus it was suggested in 3GPP to be specified as a mandatory AQM functionality in the UE, but it was not accepted. As a result, there is no guarantee that all the UEs will have AQM implemented. That means though the UEs with an advanced AQM algorithm implemented may have improved performance, for those UEs without AQM, the performance may be still bad. Thus, it is desirable to develop an AQM algorithm which is implemented in the network side to control the UEs' queue remotely. By this way, all the UEs should have enhanced performance for uplink transmission, no matter if any AQM is implemented to themselves. In this thesis, the algorithm implemented in the eNodeB is called as Receiver-AQM (R-AQM). As a comparison, the delay-based algorithm studied in [4] is called as T-AQM in the following text.

As discussed above, the main target of the R-AQM is to maintain both good link utilization and small end-to-end delay for the LTE uplink, even when the bandwidth varies. Additionally, the algorithm should be easy to implement, i.e. not having too much calculation and memory consumption.

In [4], the advantages of the delay-based AQM algorithm are shown. Comparing to a queue-size-based approach e.g. PDPC, it maintains both higher link utilization and lower end-to-end delay. Besides, it is less affected by the bandwidth variation. Thus, in the new algorithm proposed in this thesis, the delay-based principle is retained. The key point of the delay-based AQM is to consider the time of a packet stored in a buffer as an indicator of congestion. Hence, in order to implement the algorithm, the information on how long time a packet is stored in the buffer, so-called queuing delay, is required. Nevertheless, from the eNodeB side, it is impossible to know the UE's queuing delay directly without specific messaging specified for that. Therefore, some method is required to estimate the packet queuing delay.

The rest of Chapter 5 comprises three sections; Section 5.2 introduces a method to estimate the queuing delay of the headmost data in a queue by monitoring the queue length discontinuously as well as monitoring the amount of data served during each monitoring time interval. Section 5.3 presents how this method can be implemented in eNodeB to estimate the queuing delay in UE's buffer. Section 5.4 proposed the full algorithm of R-AQM.

5.2. A method to estimate queuing delay

In this section, a method to calculate the approximate queuing delay of the headmost data in a queue is proposed. Before going to details of the method, we make some assumptions. First, assume that there is a buffer with infinite capacity. Second, the queue length can be monitored only discontinuously, i.e. we can only know the queue length with discrete time intervals. Third, the length of the monitoring time interval is known and supposed to be quite small. Fourth, the amount of data served by the buffer during each time interval is also available. For convenience, in the following text, we use t_i to denote the time when the queue is monitored, while the queue length at t_i is denoted by Q_i. Additionally, the time interval between t_{i-1} and t_i is denoted by Δt_i, and the data served during time interval Δt_i is denoted as L_i.

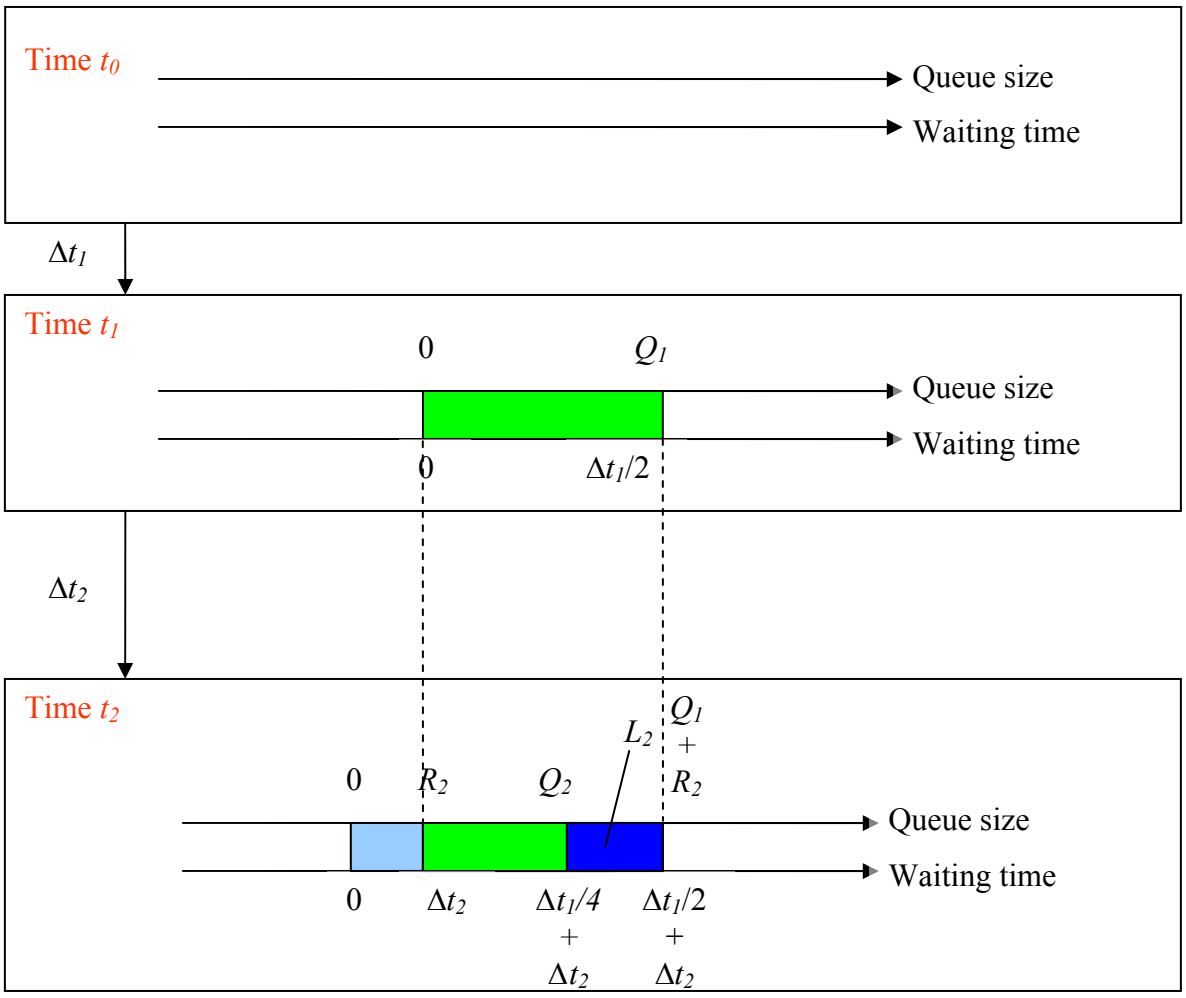

Figure 5.1: *Estimate delay at time t_0, t_1, and t_2*

Figure 5.1 shows the delay estimation procedure at time t_0, t_1 *and* t_2. Obviously, when the queue length is zero, there is no data in the buffer and the delay is also zero, like the Figure 5.1 shows at t_0. Then after a time interval Δt_1, the queue length is assumed to be Q_1. We cannot know exactly when this data came to the buffer; it can be any time between t_0 and t_1; besides, the arrival time is supposed to be uniformly distributed between t_0 and t_1, thus we can obtain that the expected queuing delay of headmost data is:

$$(t_1 - t_0) / 2 = \Delta t_1/2.$$

On the other hand, the queuing delay of backmost packet is zero, as illustrated in Figure 5.1 at t_1. Notice that there are two axes in each box, the above one denotes the queue length and below one queuing time.

After another interval Δt_2 from t_1, there are two possible situations; one is the buffer is empty again. In this case we are back to situation at t_0. The other possibility is that there is still a queue in the buffer,

but comparing to the queue at t_1, some old data may have left and new data may have come into the queue. Firstly we consider a special case: the buffer only has income R_2 but no outgo. Hence the delay of headmost data is $\Delta t_1/2+\Delta t_2$ and the leftmost 0. It is also important to notice that the data that was leftmost at t_1 is now with delay Δt_2. Thus the queuing delay of headmost data now is:

$$D_2 = \Delta t_1/2+\Delta t_2.$$

where D_2 denotes the delay of headmost data at t_2. Then we consider the queuing delay is if there is data outgo, assuming the amount of it is L_2. This situation can be treated as cutting and removing the headmost data, whose size is L_2. Obviously $L_2 < Q_1 + R_2$, otherwise the buffer is empty and the process returns to the situation at t_0. However, there are still two possible conditions, one is $L_2 < Q_1$ and the other is $Q_1 < L_2 < Q_1 + R_2$. We will investigate these two conditions respectively.

If $L_2 < Q_1$, which means all the data in the queue at t_1 has not yet removed, like illustrated in Figure 5.1 at t_2, the queuing delay of headmost data is between Δt_2 and $\Delta t_1/2+\Delta t_2$. To facilitate the calculation, also recall that Δt_2 is assumed to be quite small; the delay is then calculated by

$$D_2 = (\Delta t_2 + \Delta t_1/2 + \Delta t_2) / 2 = \Delta t_2 + \Delta t_1/4,$$

On the other hand, if $Q_1 < L_2 < Q_1 + R_2$, it indicates that not only the data at t_1 has been removed, but also some part of new income R_2 has been removed. Thus, the maximum delay then can be calculated by

$$D_2 = (0 + \Delta t_2) / 2 = \Delta t_2 / 2.$$

It is also importance to point out that, according to the assumption at the beginning of this section, the Q_2 and L_2 are known but R_2 is not. However, R_2 can be calculated using Q_1, Q_2 and L_2. Notice that the remained queue length Q_2 in Step 2 is the queue length at t_1, which is denoted by Q_1, plus the new data R_2 and minus the outgo L_2, i.e.,

$$Q_2 = Q_1 + R_2 - L_2.$$

Thus R_2 can be calculated by

$$R_2 = Q_2 + L_2 - Q_1.$$

To summarize, the delay of headmost data at t_2 can be expressed in terms of Q_1, Q_2 and L_2:

$$D_2 = \begin{cases} \Delta t_2 + \Delta t_1 / 4, & \text{if } L_2 < Q_1, \\ \Delta t_2 / 2, & \text{if } L_2 > Q_1. \end{cases} \tag{5.1}$$

Additionally, it should be noted that there are several values that should be stored for future calculation, i.e., R_2, Q_2 and the corresponding queuing delays Δt_2 and D_2 (which is $\Delta t_2 + \Delta t_1 / 4$ in the example illustrated in Figure 5.1).

The procedure continues step by step in the same way as explained above. Next we will consider a more general situation: how to calculate the delay of headmost data at t_n. As illustrated in Figure 5.2, every new income R_i during time interval Δt_i is calculated and recorded until it is removed from the queue. This can be done by maintaining a data structure, e.g. an array, to record received bits in every time interval. New data R_i can be calculated by monitoring the queue length and outgoing bits, i.e.

$$R_i = Q_i + L_i - Q_{i-1}, \tag{5.2}$$

where Q_i, L_i, Q_{i-1} are known variables. Using Equation (5.2) all the values R_i can be calculated; additionally, they are recorded with their corresponding monitoring intervals, i.e. Δt_i, as showed in Figure 5.2. The light blue box indicates the new income R_n, while each green one denotes the received data in the previous time intervals. Yellow one, on the other hand, presents the headmost data. However, some part of it has been removed during time interval Δt_{i-1}. Thus, the queuing delay of headmost data, if we assume that anything is not removed yet, like Figure 5.2 (a) shows, should be

$$(\sum_{i=r}^{n} \Delta t_i + \sum_{i=r-1}^{n} \Delta t_i) / 2,$$

where r is the index of monitoring event t_r when the headmost data, i.e. the oldest data still existing in the queue, has arrived. However, the fact is that L_n data is already removed from the queue. Thus the actual headmost data is the point Q_n, which means that the queuing the delay of headmost data is dependent on which area the Q_n is located. For instance, assume that the Q_n is between $\sum_{i=r+1}^{n} R_i$ and $\sum_{i=r}^{n} R_i$, then the max queuing delay is calculated as the average of their corresponding delays, i.e.,

$$D_n = (\sum_{i=r+1}^{n} \Delta t_i + \sum_{i=r}^{n} \Delta t_i) / 2,$$

just as Figure 5.2 (b) shows. Furthermore, D_n is stored in the data structure, as well as its corresponding queue length Q_n. Additionally, in order to facilitate future calculation and memory consumption, the elements indicating those removed data should be discarded from the data structure, i.e., $\sum_{i=r}^{n} R_i$, $Q_{n-1} + R_n$ and their corresponding delays.

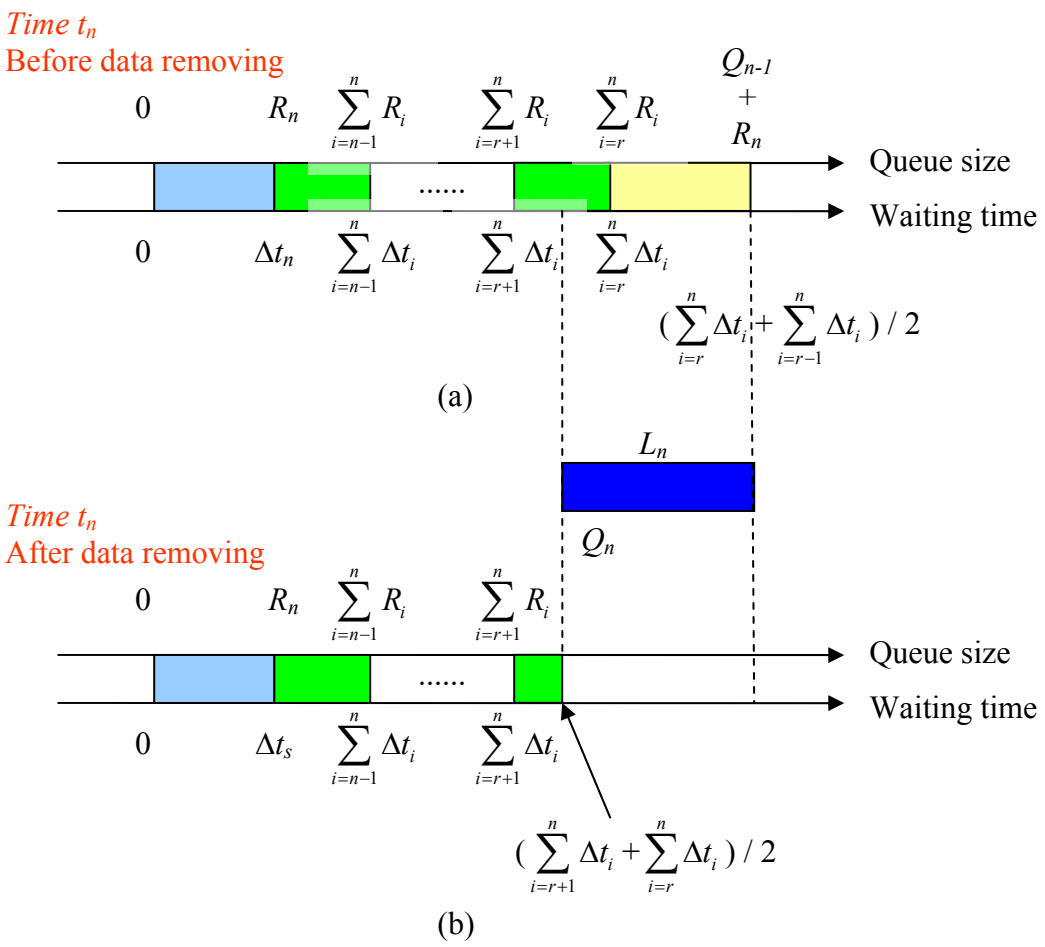

Figure 5.2: *Estimate delay at t_n*

By the method described above, we can calculate the queuing delay of the headmost data in every time interval. The pseudo code of method is given in Algorithm 5.2. As seen from Figure 5.2, in order to store the amounts of bits received in every time interval and their corresponding queuing delay, i.e. the sum of R_j, and Δt_j, arrays or similar data structure are required. In the Algorithm 5.2, *R[]* and *D[]* denote the $\sum_{i=l}^{n} R_i$ and $\sum_{i=l}^{n} \Delta t_i$ respectively; furthermore, *R[].addFirst(arg)* means attaching the argument to the head of *R[],* similarly, *R[].addLast(arg)* means attaching the argument to the end of *R[]*.

At any time t_i, the queue length Q_i, the amount of data served by the buffer L_i, and time interval Δt_i, are assumed to be available. Besides, *lastIndex* denotes the index of last element in array.

Algorithm 5.2: packet delay estimation

```
initial Array D[]
initial Array R[]
Q_old← 0
Each time the queue length Q is monitored and also L and Δt available:
    if (Q = 0)
        initialize D[]
        initial R[]
        Q_old ← 0
    else
        delay ← 0
        for (j ← lastIndex to 0)
            if (R[j] > Q_old – L)
                delay ← D[j]
                remove R[j] from R[]
                remove D[j] from D[]
            else
                if (R[j] + Q + L – Q_old < 0 )
                    remove element from R[] and D[]
                else
                    R[j] ← R[j] + Q + L –Q_old
                    D[j] ← D[j] + Δt
        estimatedDelay ← (delay + Δt + D[lastIndex])/2
        Q_old ← Q
        R[].addFirst(0)
        R[].addLast(Q)
        D[].addFirst(0)
        D[].addLast(estimatedDelay)
```

5.3. Estimation of delay in eNodeB

In Section 5.2 a method to estimate the queuing delay of headmost data is introduced. To enable this, the queue length should be able to be monitored with small time intervals; additionally, both the length of each time interval and the amount of bits served by the buffer in each time interval are also required. In this section it is presented how this method to estimate the queuing delay of the buffer in UE is implemented in eNodeB. As Section 2.4.3 introduced, BSRs are transmitted from the UE to inform the eNodeB the size of the queue in the buffer of the UE. BSRs are triggered by some special events which are presented in Section 2.5 as well as in [23]. Typically, the triggering interval is around tens of milliseconds. Thus, the BSR can be seen as a monitor of the queue length with a small time interval. If we can obtain the length of time intervals and the amount of bits served by the buffer in each time interval, the queuing delay of headmost data can be estimated by Algorithm 5.2.

To investigate the problem further, firstly we assume that at time t_{i-1} a BSR is triggered and transmitted together with a MAC PDU. After a time interval Δt_i the next BSR is triggered at time t_i. Let's treat the data in transmission as a stream, thus the data between these two BSRs is exactly the RLC SDUs served by the buffer, in addition to the MAC headers and CRC codes. Then we assume these two BSRs are received by the eNodeB at r_{i-1} and r_i respectively, with time interval $\Delta t_i' = r_i - r_{i-1}$. At the same time, the data between them is also received during r_{i-1} and r_i. If no HARQ retransmissions are not performed, the $\Delta t_i'$ should be equal to the Δt_i; furthermore, the amount of data delivered to higher layer during r_{i-1} and r_i should approximate the data amount served by the UE's buffer during t_{i-1} and t_i. In addition to the queue length Q_{i-1} and Q_i indicated by these BSRs, the queuing delay of headmost data can be calculated.

However, this is only the error-free situation. If there are some HARQ errors detected and retransmissions are required, the situation is more complicated. First, the Δt_i and $\Delta t_i'$ may have different values if the retransmitted MAC PDU includes the BSR. Second, due to the in-sequence delivery function provided by the RLC layer, the received RLC SDU will not be directly delivered to higher layer if previous SDUs are not correctly received; consequently, the delivered data during r_{i-1} *and* r_i in eNodeB may quite differ from served data in UE during t_{i-1} *and* t_i.

As discussed above, some restrictions will affect the accuracy of the delay estimation. Due to the complexity in realistic situations, we can only investigate it in a statistical way e.g. transmitting large amount of data and see how close the estimated delay and the real delay are. Thus, the performance of the estimation s will be investigated by simulations with different fixed HARQ error probabilities in Section 7.1.

5.4. Delay-based AQM in eNodeB

In section 5.2 and 5.3, a method to estimate the queuing delay has been proposed. After that, this new method is proposed to be implemented in eNodeB to estimate the delay of the queue in a UE. Implementing the delay-based AQM algorithm in eNodeB is quite intuitive. We just replace the accurate packet delay by the estimated delay, as well as we replace the accurate queue length by the BSR reported size. Nevertheless, the maxAgeThreshold is removed from the T-AQM algorithm. In T-AQM, the maxAgeThreshold is used to discard the packets whose queuing delay exceeds the maxAgeThreshold, without considering the minInterDropTime. For T-AQM, this is quite a useful mechanism which can drain the buffer quickly when there is significant bandwidth down-switch [4]. However, the mechanism is based on the fact that the T-AQM is implemented in UE side directly operating the queue. For R-AQM, there are some differences. The R-AQM is implemented in eNodeB and thus cannot directly operate the queue in a UE. Thus, even the eNodeB can correctly estimate the queuing delay of received packets and drop all the packets whose queuing delay exceeds the maxAgeThreshold, this is not much help for draining the UE buffer fast, because the packets are not dropped directly from the queue. Additionally, if a large amount of packets are dropped after they have been transmitted through the physical channels, the allocated physical resources to transmit them are actually wasted. Hence, it is not wise to us to define maxAgeThreshold in R-AQM. To handle the situation when there is significant bandwidth down-switch, a possible solution is combining the R-AQM with PDCP discard [23]. A more detailed investigation on the combination will be presented in Section 7.3.2.

As discussed above, the general algorithm of R-AQM is:

Algorithm 5.3

```
if (bsrSize < lowerDropThreshold)
    deliver packet

else if ( estimated_delay > minAgeThreshold AND
        now – lastDropTime > minInterDropTime )
    drop packet
    lastDropTime ← now

else
    deliver packet
```

The bsrSize is the queue size reported by a BSR and estimated_delay is the delay estimated when the BSR is received. LastDropTime is the time when last packet is discarded.

The detailed algorithm flow is illustrated in Figure 5.3. There are two separate procedures: one is estimating the maximum queue delay and the other is deciding if the packet drop will be triggered. However, they are also interactive with each other, e.g. in order to calculate the maximum queue delay, the information on how much bits is received during the interval of two received BSRs is provided by the other procedure. On the other hand, the decision of packet drop is based on the calculations of the first procedure.

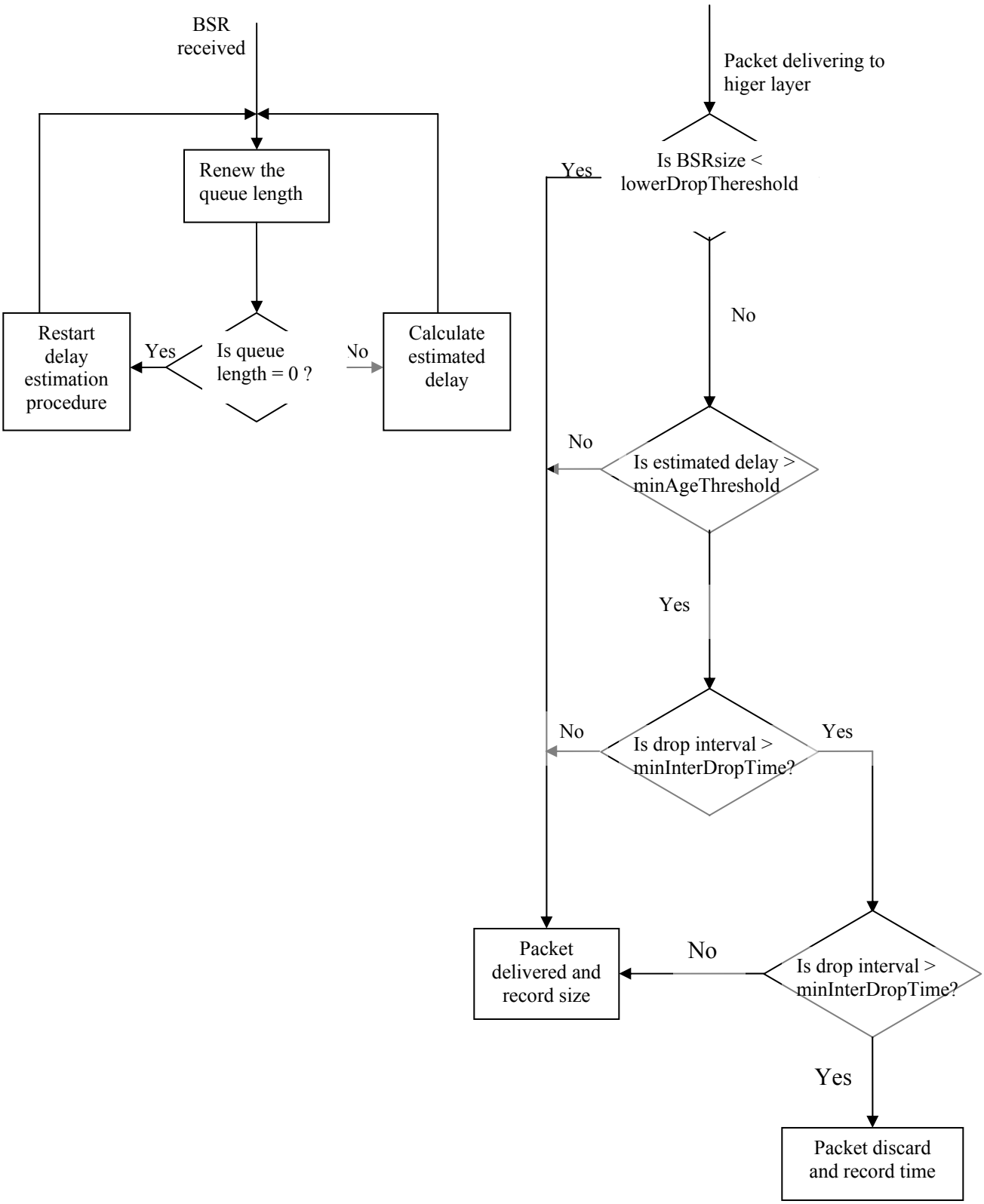

Figure 5.3: *Detail R-AQM algorithm flow chart*

6. Simulator configurations

All the results in this thesis are obtained from the simulations performed by the simulator developed by Ericsson Research. It includes a large number of Java classes which specify the details of the radio network, transport network, propagation models, Internet etc. In addition, most of the parameters in this simulator are configurable, which makes the simulator a convenient and powerful tool for radio network simulations. In this thesis, a fully implemented LTE system is used to investigate the performance of AQM algorithms.

6.1. Simulation Models

The R-AQM algorithm discussed in Chapter 5 is implemented in the RLC layer of eNodeB in the simulator. Considering the ARQ functionality provided by RLC layer, which will guarantee the error-free transmission, the dropping performed by AQM happens when RLC SDUs are being delivered to higher layers, as Figure 6.1 shows.

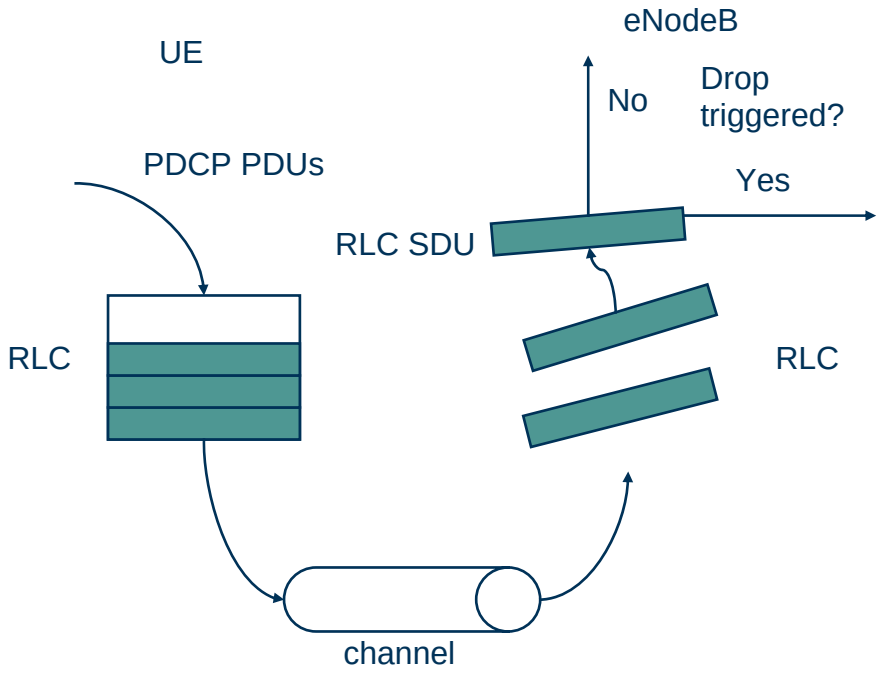

Figure 6.1: *Simulation model of R-AQM*

As comparison, performance of other Passive Queue Management and Active Queue Management algorithms for LTE uplink will also be investigated. The algorithms include PDCP discard specified in [23], drop-from-front and T-AQM proposed in [4]. Unlike the R-AQM, these three algorithms are supposed to be implemented in UE. It is important to be noted that the PDCP discard is supposed to be implemented in the PDCP layer and discard the PDCP SDU. However, in order to simplify the

configuration of the simulator, the RLC SDUs are discarded in all simulations. The simulation model of these three algorithms is illustrated in Figure 6.2. For each outgoing RLC SDU, the algorithms will check if the drop is triggered; if yes, the RLC SDU will be discarded, otherwise the SDU is attached by a RLC head and delivered to MAC layer. The algorithms of PDCP discard, drop-from-front and T-AQM are referred as Algorithm 5.1, Algorithm 6.1 and Algorithm 4.2 respectively.

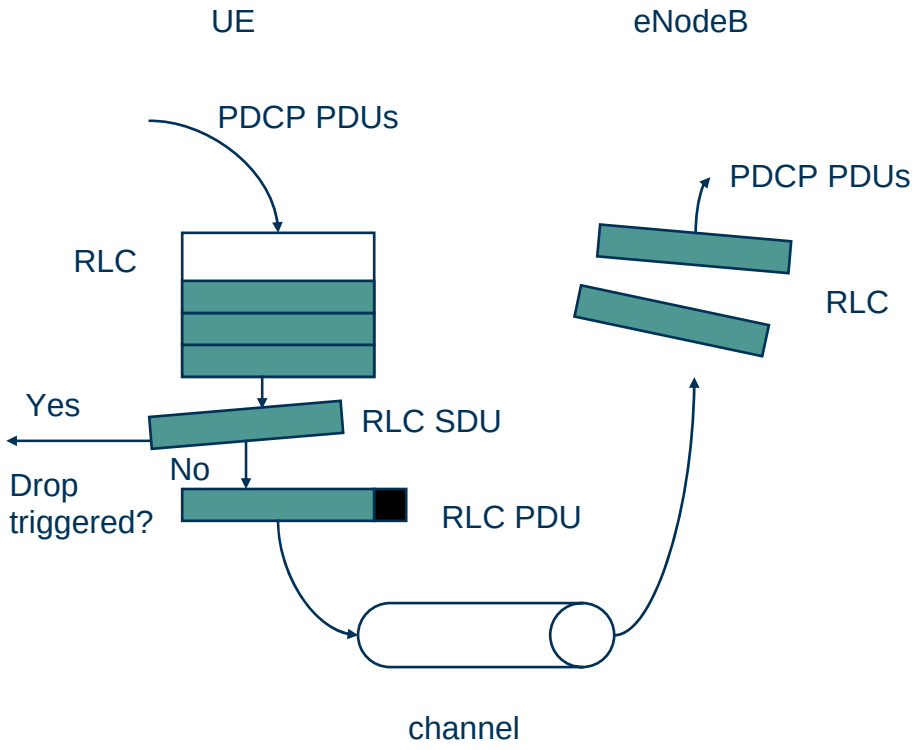

Figure 6.2: *Simulation model of T-AQM, PDCP discard and drop-from-front*

Algorithm 6.1: Drop-front-front
for each outgoing packet **if** (queue_size > bufferLimit) discard packet **else** transmit packet

6.2. Parameter settings

In the simulator, there are a large number of parameters that can be configured. However, in the simulations which will be done in Chapter 7, we leave most of them as default values. Some important parameters are listed here:

- Max *cwnd*: 2097 Kbytes
- Buffer size: 1.25 Mbytes
- Transport network delay (one way): 20 ms
- Internet delay (one way): 40 ms
- TTI duration: 1 ms
- BSR period: 20 ms
- Number of Base Station: 1
- Number of downlink beams per cell: 1
- Cell radius: 500 m
- No MIMO used

It is important to be noted that in the subsections of Chapter 7, if there is no obvious indication, the parameters are set as the values listed above. Otherwise the changed parameters will be listed in those subsections.

7. Simulation results and analysis

Chapter 5 proposed a method to estimate the queuing delay of UE in eNodeB as well as it proposed an R- AQM algorithm. In this chapter more detailed investigation on the delay estimation method will be done using simulations. Additionally, the performance of R-AQM with different thresholds will be investigated and some optimal values of these parameters will be proposed. Finally, the comparison of different queue management mechanisms, i.e. R-AQM, T-AQM [4], PDCP discard [23] will be done based on the simulations. This chapter is divided into three parts: Section 7.1 illustrates the performance of delay estimation with different HARQ error probability; Section 7.2 presents the selection of optimal threshold for the R-AQM; while Section 7.3 compares the performance of different queue management mechanisms.

7.1. Performance of delay estimation in eNodeB

Sections 5.2 and 5.3 introduced a method to estimate the queuing delay and how it is implemented in the eNodeB. Those sections also presented that error may happen when HARQ error is detected due to the retransmission and in-sequence delivery function provided by RLC layer. To investigate how the HAQR error affects the estimated delay, a few simulations are done with manually fixing the HARQ error probability to 0%, 10%, 30% and 50%. Additionally, no AQM is implemented in these simulations. The method to handle the RLC transmitter buffer is drop-from-front. Some related parameters are set as:

- Buffer limit: 375 Kbytes
- File size: 20 Mbytes
- Number of File transfers: 1

Figure 7.1 illustrates the estimated delay, which is denoted by the red points, as well as the real queuing delay, which is denotes by the green points. Additionally, the x-axis denotes the time and y-axis denotes the delay. From the figures it can be observed that when the HARQ error probability increases, the errors of delay estimation increase as well. However, it is to be noted that our purpose of delay estimation is to serve the R-AQM that trigger the packets drop, i.e. when the estimated delay exceeds the minAgeThreshold, a packet may be discarded. From all the figures we can see that even though there are estimation errors, the estimated delay is good enough to trigger the packet drop. This is because on one hand, the estimated delay follows the real delay quite well in a statistical way if we ignore the erroneous estimated values. On the other hand, the erroneous estimated values are usually below the real value, thus there should be few unexpected packet drops by erroneous estimation.

As discussed above, the delay estimation method works well even when the HARQ error probability is up to 50%, if our purpose is to trigger the packet drop. It is also worthy to point out that in realistic radio network, the HARQ error probability should be much less than 50%. Hence, we can conclude that the delay estimation method is qualified to serve the R-AQM.

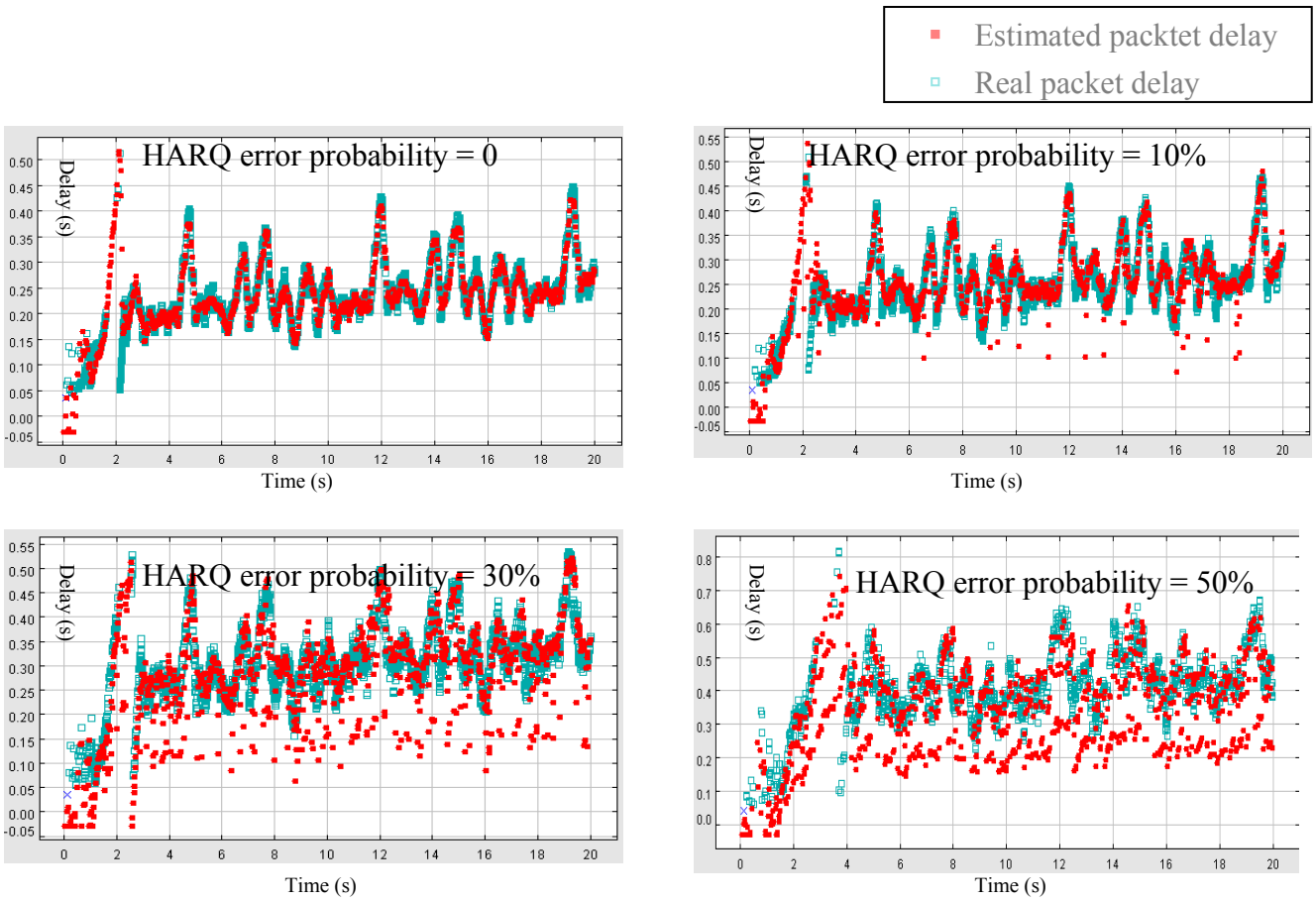

Figure 7.1: *Delay estimation with different HARQ error probabilities*

7.2. Parameters selection

Section 7.1 has showed that the delay estimation method works well to serve for the R-AQM. Based on simulation results, this section proposes some proper values for different parameters of the R-AQM.

7.2.1. Selection of minAgeThreshold

The minAgeThreshold is the most important parameter because it determines how large the estimated delay can be before packet drops are done. As presented in [4], to maintain full link utilization in fixed bandwidth, the minAgeThreshold should be at least RTT_{e2e}. In this section a number of simulations will be done to investigate the performance of R-AQM with different values of minAgeThreshold. It should be noticed that in these simulations the lowerDropThreshold is disabled because no reasonable value is available until now. However, the minInterDropTime is set to be 0.3 s. The reason for the setting of minInterDropTime is that if we disable the minInterDropTime, the consecutive drops of packets degrade the throughput and it is difficult to obtain meaningful results from the simulations, especially when the bandwidth varies. The selection of minInterDropTime to 0.3 s is quite arbitrary, but a more detailed discussion about the selection of it is presented in Section 7.2.2.

The related settings of the simulations are:

- File size: 10 Mbytes
- Number of file transfers: 10

Firstly the throughputs in a fixed bandwidth scenario are illustrated in Figure 7.2. From the figure a trend can be observed: At the beginning, the throughput increased when a minAgeThresholsd is increased. The reason is that when the minAgeThreshold is quite small, the packets will be discarded before the PC is fulfilled, which will cause throughput degradation. However, when the thresholds exceed some value, the throughputs have no obvious improvement any more. That is because when the threshold is large enough, the PC is fulfilled and increasing minAgeThreshold can only enlarge the queue length as well as end-to-end delay. Thus, an optimal value of minAgeThreshold is supposed to be the value that maintains the throughput high but the end-to-end delay is small.

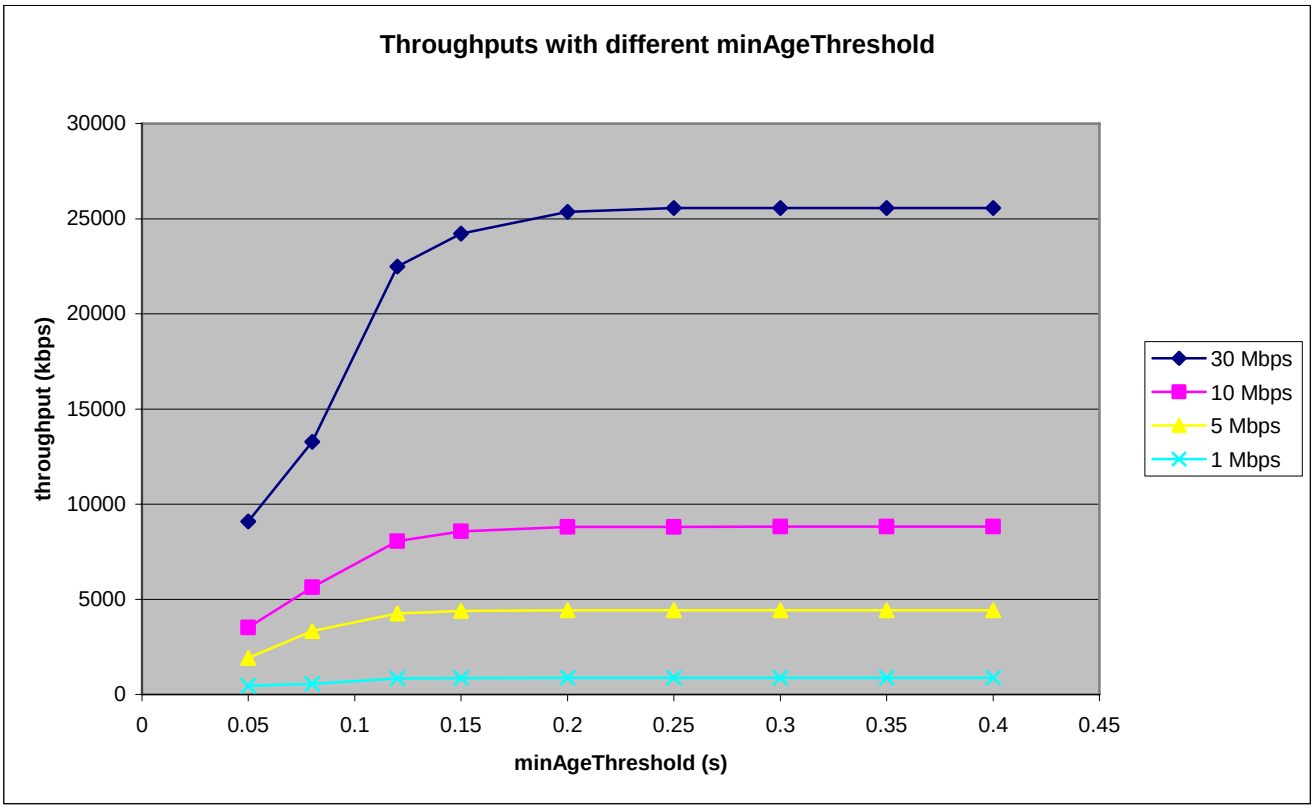

Figure 7.2: *Throughput with different bandwidths and minAgeThresholds*

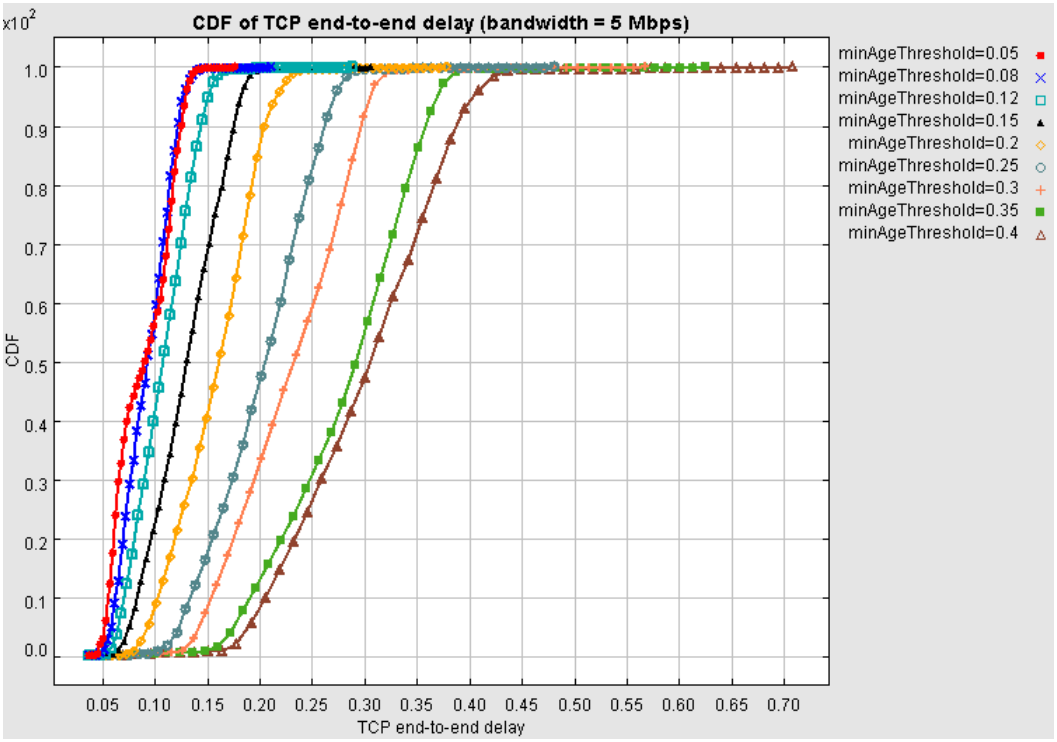

Figure 7.3: *CDF of TCP end-to-end delay with different minAgeThreshold when bandwidth is 5 Mbps*

Figure 7.3 illustrates the TCP end-to-end delays with bandwidth of 5 Mbps. From the curve of 5 Mbps in Figure 7.2 we can see that when the minAgeThreshold exceeds 0.15 s, there is no improvement when increasing the minAgeThreshold. On the other hand, the TCP end-to-end delay increases with less strict minAgeThreshold all the time, as showed in Figure 7.3. Similar observation can be found with other bandwidths. Thus, 0.15 s is a good value for minAgeThreshold when the bandwidth is fixed. However, it is important to point out that some disturbances can be observed from the curve in Figure 7.4, where the bandwidth is 128 kbps. To look into this problem, the numbers of RTOs are showed in Table 7.1 with different minAgeThresholds. As a comparison, the numbers of RTOs with 1 Mbps bandwidth are also listed.

Table 7.1: Times of RTOs with different minAgeThreshold

minAgeThreshold (s)	0.05	0.08	0.12	0.15	0.2	0.25	0.3	0.35	0.4
Times of RTOs (128 kbps)	207	90	59	62	80	63	64	49	58
Times of RTOs (1 Mbps)	1	7	0	0	1	0	0	0	0

From Table 7.1 it can be seen that when the minAgeThreshold is 0.2 s and the big throughput degradation can be observed in Figure 7.4, the times of RTOs are higher than e.g. when minAgeThreshold is 0.15 or 0.25. Additionally, when the bandwidth is 128 kbps, the total number of RTOs is quite high even the minAgeThreshold is set to 0.4, comparing to the number of RTOs with bandwidth 1 Mbps. The result indicates that a packet drop in the scenario of low bandwidth have higher possibility to cause RTO than in high bandwidth. Too many RTOs will definitely degrade the throughput of TCP significantly. Thus, the parameter lowerDropThreshold is created to prevent from

the packet drop when the queue is quite small. More detailed study on lowerDropThreshold will be in Section 7.2.3

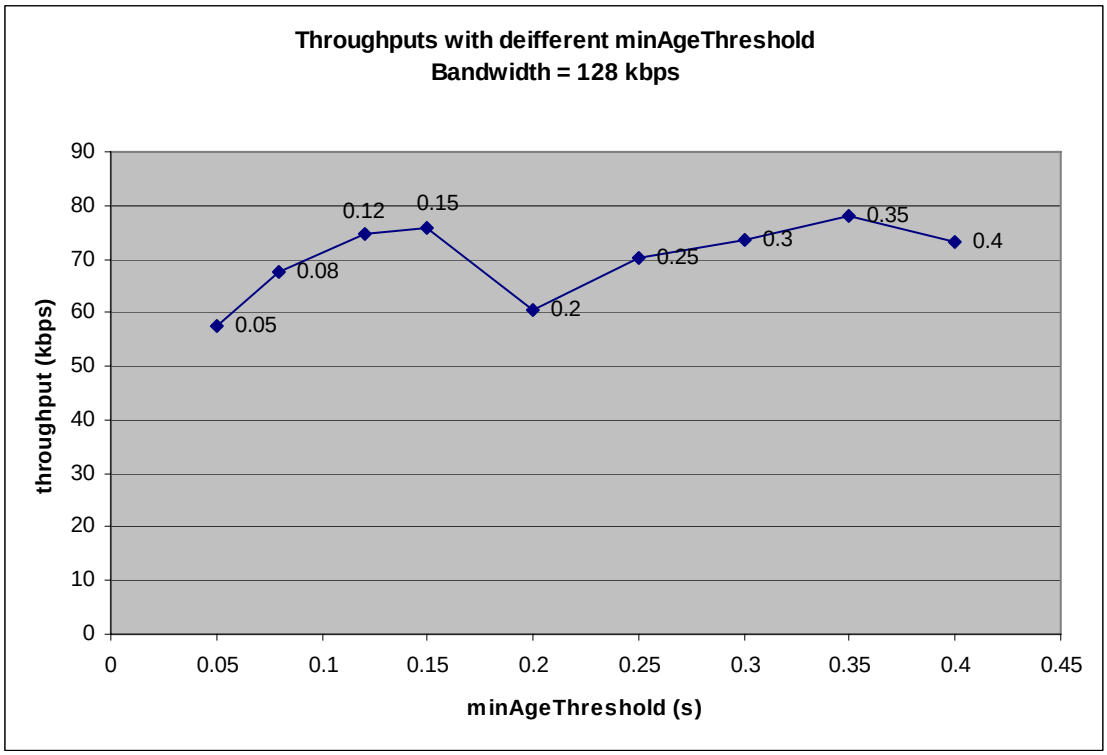

Figure 7.4: *Throughput with different minAgeThreshold when bandwidth is 128 kbps*

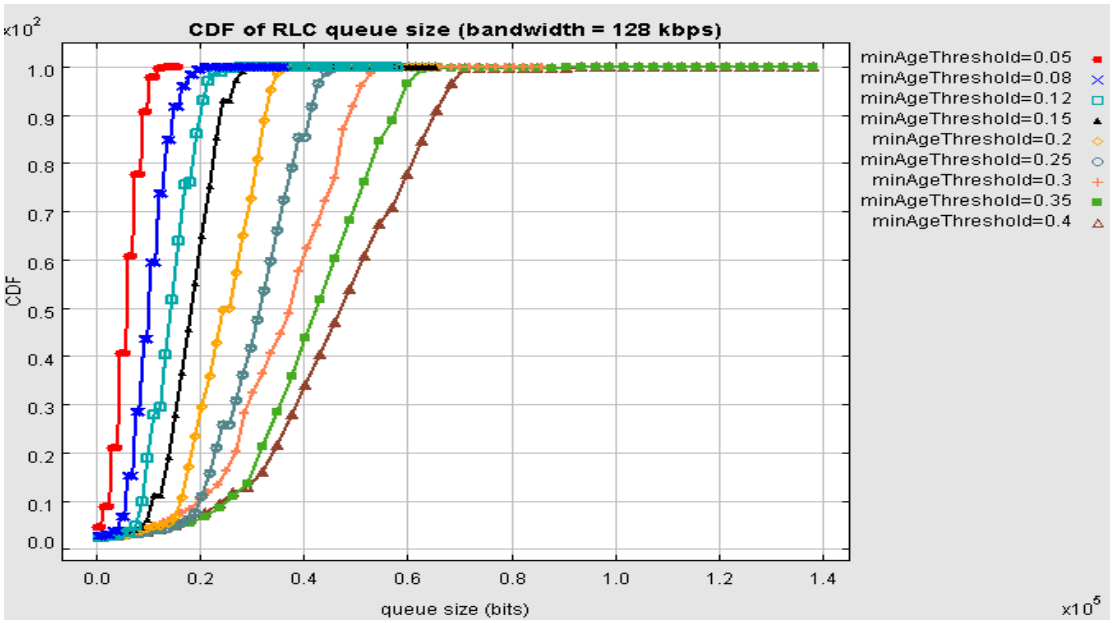

Figure 7.5: *CDF of queue size with different minAgeThreshold when the bandwidth is 128 kbps*

Next we will investigate the performance of different minAgeThresholds in varying bandwidth scenario. We consider not only deterministic changes but also a realistic radio network scenario where the bit rate changes due to the physical environment. Figure 7.6 and Figure 7.7 illustrate the throughput of deterministic changing bandwidth. Figure 7.6 shows the throughput when the bandwidth oscillates between 1 Mbps and 20 Mbps with period 20 second while Figure 7.7 shows the throughput when the bandwidth oscillates between 512 kbps and 5 Mbps with period 5 second. It is important to point out that, as found in [4], a quite low variation, e.g., high bandwidth is a few times of low bandwidth or quite fast variation, e.g., the oscillation period is less than 1 s, the performance in these situations is quite similar to the fixed bandwidth. Hence, in this section, only large variations with period 5 s and 20 s are investigated.

From the Figure 7.6 a similar dependency of throughput and minAgeThreshold can be observed as with the fixed bandwidth. Nevertheless, the "reasonable value" in this situation is around 0.2 s, which is a little bit more than in a fixed bandwidth. This example indicates that larger minAgeThreshold may obtain better robustness against bandwidth variations; on the other hand, larger minAgeThreshold means a larger end-to-end delay. Thus, there is again a tradeoff between the throughput and the delay.

From the Figure 7.7 no obvious trend of throughout as a function of minAgeThreshold can be observed. When the minAgeThreshold is 0.05 s and 0.08 s, the throughput is very bad; the throughput is even much less than 100 kbps. After that, the throughput improves significantly when minAgeThreshold increases from 0.08 s to 0.15 s. Then throughput degradation happens again when the minAgeThreshold is 0.25 s and 0.3 s. We investigate this problem by listing the number of RTOs with different minAgeThresholds in Table 7.2.

Table 7.2: Times of RTOs with different minAgeThresholds when bandwidth varies

minAgeThreshold (s)	0.05	0.08	0.12	0.15	0.2	0.25	0.3	0.35	0.4
Times of RTOs	206	211	83	25	14	153	91	16	17

From Table 7.2 we can see that those minAgeThresholds that had throughput degradation has a large number of RTOs. The reason is that, even we have set the minInterDropTime to 0.3 s, it is not enough to prevent the occurrences of multiple drops in one window, especially when the bandwidth down-switches e.g. from 5 Mbps to 512 kbps. Increasing the minInterDropTime can reduce the RTOs and improve the throughput. More detailed investigation on minInterDropTime will be in Section 7.2.2.

Additionally, it should be noticed that the down-switched bandwidth may cause a delay burst problem, which means in a short period, the queuing delay may be quite large. The reason is, when the bandwidth is high, there is a large queue in the buffer. Once the bandwidth down-switches to a lower lever, the PC decreases. The queue cannot adapt to that very fast, thus some packets in the queue will get large queuing delay. In the T-AQM studied in [4], this problem is handled by discarding all the packets whose queuing delay exceeds the maxAgeThreshold. However, for the R-AQM, it cannot handle this situation, as discussed in Section 5.4.

One scenario throughput can degrades is when the bandwidth up-switched e.g. from 512 kbps to 5 Mbps, since there is too small queue in the buffer to fulfill the suddenly inflated Pipe Capacity.

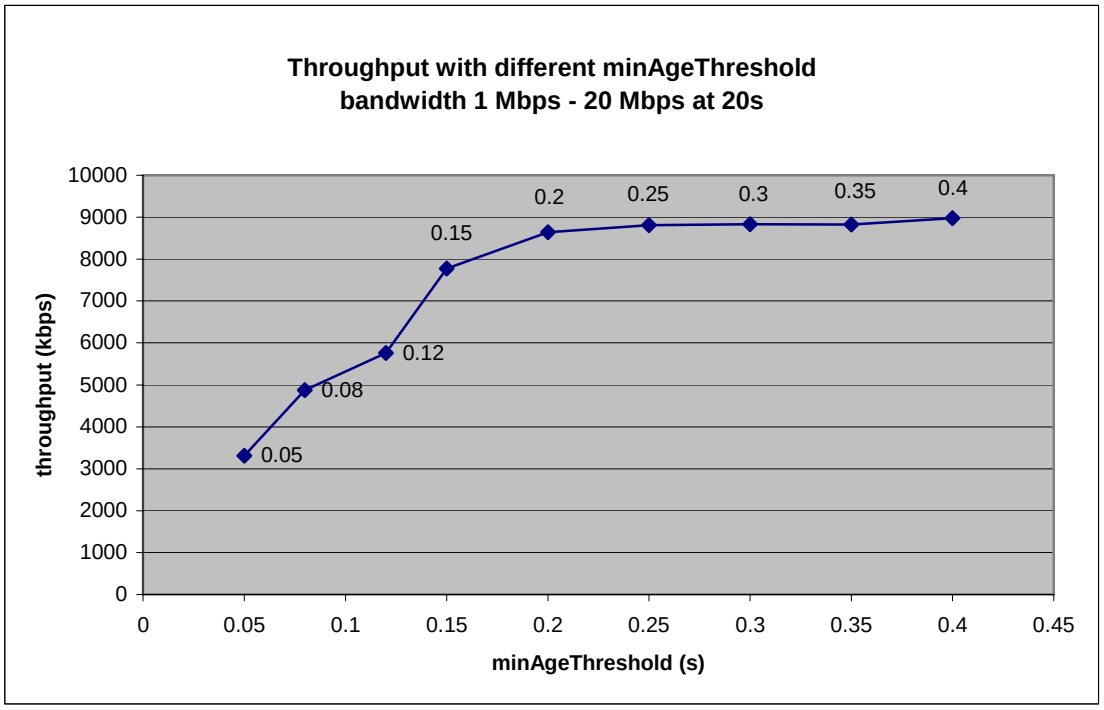

Figure 7.6: *Throughput with different minAgeThreshold when bandwidth varies*

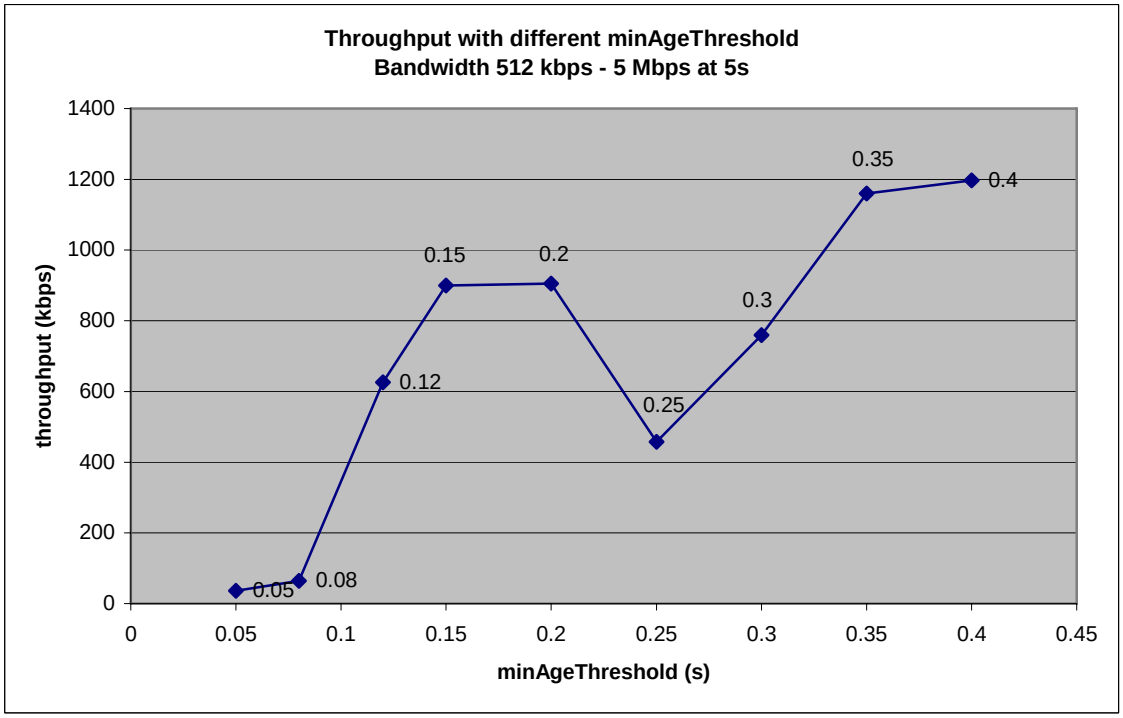

Figure 7.7: *Throughput with different minAgeThreshold when bandwidth varies*

Next, the performance with different minAgeThreshold in realistic radio network environment will be investigated. In these simulations, no predefined bandwidth is assigned and all the scheduling decisions depend on the instantaneous environments and link adaptation. The related parameter is:

- UE speed: 0.83 m/s

Figure 7.8 illustrates the throughput curve with different minAgeThresholds. From Figure 7.8 a similar trend of throughputs can be observed as with the fixed bandwidth. Also in this case the "optimal value" of minAgeThreshold is 0.2 s.

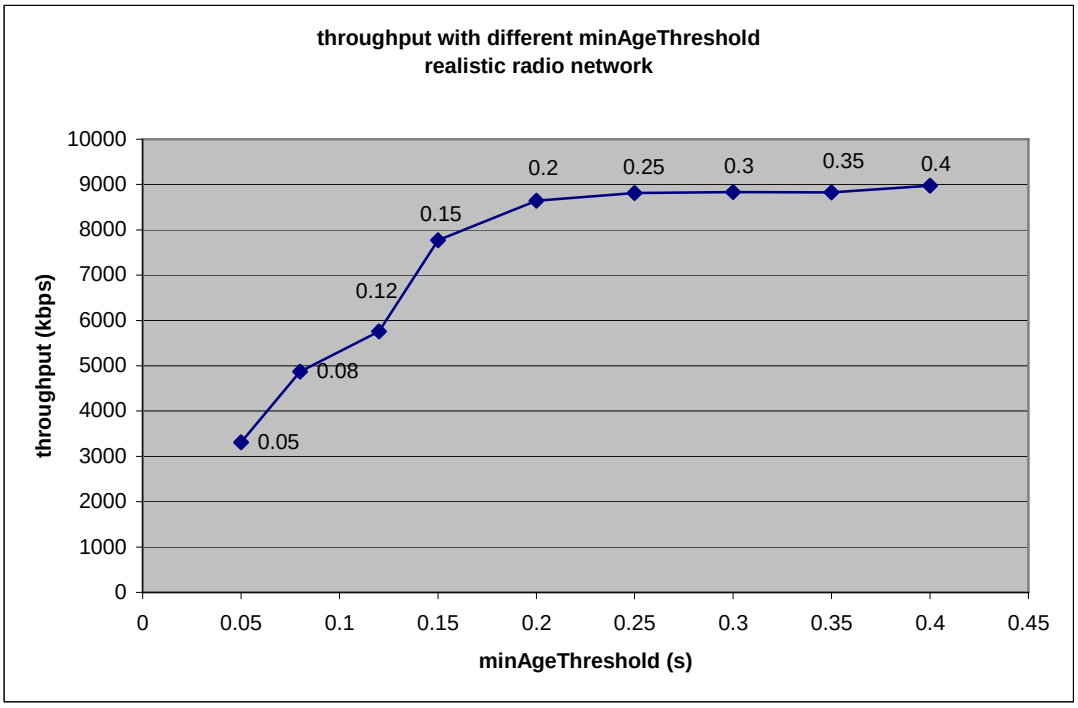

Figure 7.8: *Throughput with different minAgeThreshold in a realistic radio network*

As discussed above, we can conclude that the optimal minAgeThreshold is supposed to be around 0.2 s. It can maintain the high throughput as well as low end-to-end delay both with fixed and varying bandwidth, though the throughput has degradation when the bandwidth is 128 kbps. Later in this section we try to improve it by correctly setting the lowerDropThreshold. Additionally, when the bandwidth oscillates large during a moderate period, e.g. 5 s, a large number of RTOs is observed; this can be improved by setting the minInterDropTime, which will be discussed in the following Section 7.2.2.

7.2.2. Selection of minInterDropTime

The purpose of the parameter minInterDropTime is to prevent consecutive drops. For TCP, multiple drops in one congestion window may cause RTO which will reduce the throughput considerably. Quite a lot of research on how to improve the throughput when multiple packets drops in one window has been done and SACK is a good option for that. Comparing to Tahoe and Reno, SACK can

acknowledge separate received bunches of packets so that the possibility of RTO decreases. However, it still faces the RTO problem, especially when the retransmitted packets are discarded. Therefore, the AQM algorithms usually freeze the packet drop in a predefined period once a packet is discarded. In R-AQM, the minInterDropTime is defined to separate two continuous drops at least minInterDropTime to decrease the possibility of RTO.

The simulations in Section 7.2.1 set the minInterDropTime to 0.3 s and it seems suitable in most situations except for the oscillating bandwidth with the period of 5 s. More simulations will be done in this section with focus on the TCP performance with different minInterDropTime.

The general parameters are showed below. In addition, the minAgeThreshold is set to 0.2 s, as proposed in Section 7.2.1. The lowerDropThreshold is still disabled.

Table 7.3 shows the throughput with different minInterDropTimes and bandwidths. In addition, Figure 7.9 illustrates the times of RTOs. From them, it is obvious that when the minInterDropTime is zero, the numbers of RTOs are quite high and therefore the throughput has severe reduction. Then the throughput improves with increasing minInterDropTime. When the minInterDropTime is around 0.6 s, the number of RTOs is very small. However, the curve in the scenario in which the bandwidth oscillates between 512 kbps – 5 Mbps, varies all the time. Thus it is difficult to predict the exact performance in such a varying bandwidth environment. Additionally, as Figure 7.10 shows, the TCP end-to-end delay is quite similar, no matter which the minInterDropTime is. Thus, a value around 0.6 s is preferred. Nevertheless, as presented in [4], this value might be not quite suitable for multi-flow scenario because it may cause unfair sharing problem. Hence, there is tradeoff between obtaining a good throughput and maintaining fairness in multi-flow scenario. In the following simulations, the previous choice, i.e. obtaining good throughput is chosen and the minInterDropTime is selected as 0.6 s.

Table 7.3: Throughput with different minInterDropTime and bandwidth

	0	0.2s	0.4s	0.6s	0.7s	0.8s	1s	1.5s	2s
30 Mbps	14587.3	25357.99	25358	25358	25358	25358	25358	25358	25486
5 Mbps	3498.8	4422.42	4422.7	4422.7	4422.74	4422.74	4423	4422.7	4424
1 Mbps	244.01	787.8	806.61	869.11	869.11	869.11	869.1	869.28	869
1 – 20 Mbps at 20s	497.94	3125.65	3271.8	3006.6	3228.09	3170.66	3147	3117.1	2756
512k – 5M bps at 5s	11.43	488.85	804.88	981.43	968.07	1369.8	1004	877.07	981.6
Realistic network	2824.02	4741.2	4671	4744.9	4744.88	4744.88	4686	4656.5	4760

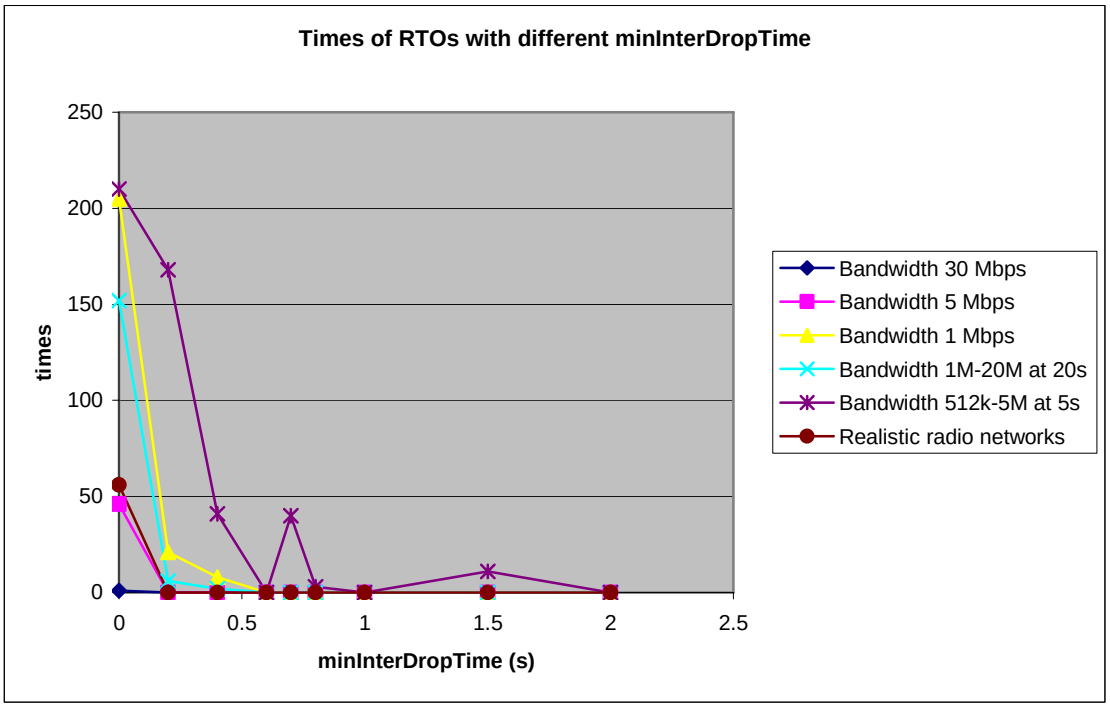

Figure 7.9: *Times of RTOs with different minInterDropTime*

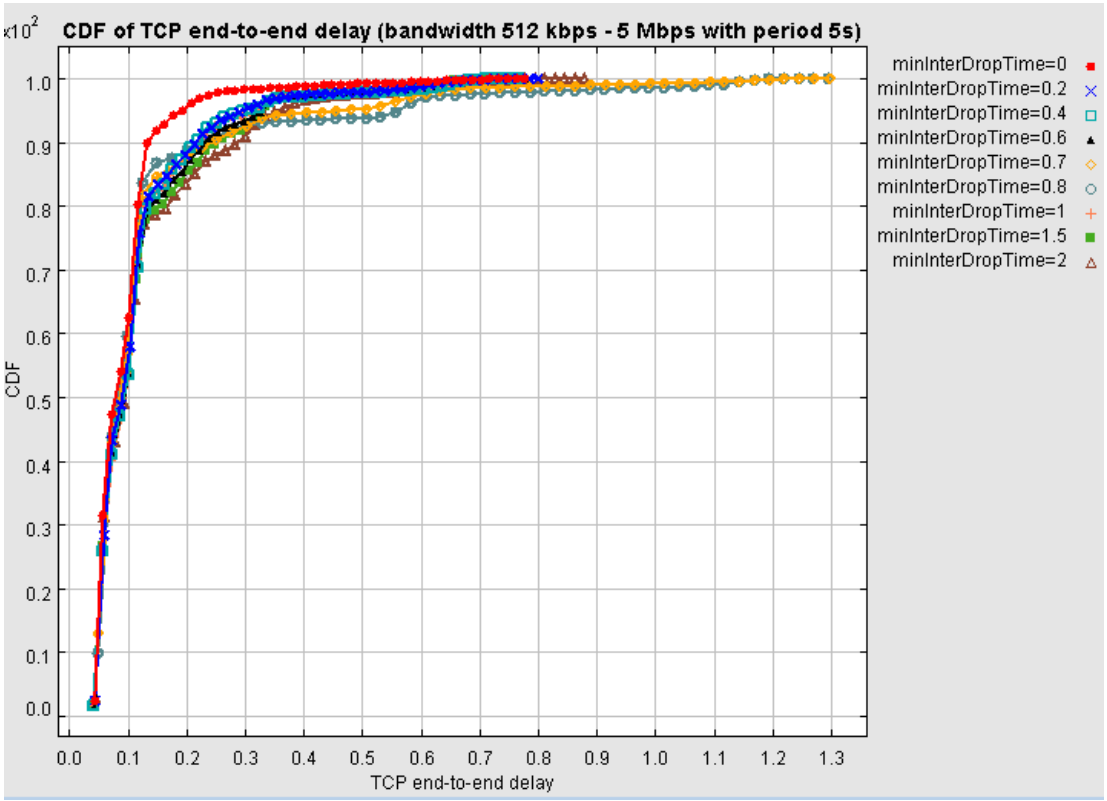

Figure 7.10: *CDF of TCP end-to-end delay with different minInterDropTime*

7.2.3. Selection of lowerDropThreshold

The purpose of lowerDropThreshold is to prevent a packet drop when the queue is quite small. The reason is that when the bandwidth is low, the transmission delay is considerable large and cannot be neglected any more. Furthermore, a packet drop in the scenario of low bandwidth have higher possibility to cause RTO than in high bandwidth, As seen from Figure 7.5 and Table 7.1 in Section 7.2.1, when the queue size is quite small e.g. less than 60 Kbits, a packet drop has a quite high probability to cause the RTO. Recall that this is the situation with bandwidth 128 kbps. In reality there might be even lower bandwidths; therefore, a larger minAgeThreshold is required to maintain a good throughput as compared to a high bandwidth scenario. However, in this thesis the minAgeThreshold is set to some fixed value for simplicity, thus the lowerDropThreshold is selected to maintain the good throughput also in a low bandwidth scenario.

As presented in [4], the lowerDropThreshold takes effect only when the bandwidth is quite low, e.g. less than 1 Mbps. Thus the simulations in this section are only with a low bandwidth. The related setting of parameters:

- File size: 2 Mbytes
- Numbers of file transfers: 5
- minAgeThreshold: 0.2 s
- minInterDropTime: 0.6 s

Figure 7.11 illustrated the throughput with different lowerDropThresholds in different bandwidth scenarios. It should be noticed that in the figure, the throughput is scaled, which means the values divided by the throughput without any packet drop. On the other hand, Table 7.4 shows the number of RTOs with the same lowerDropThreshold and bandwidth. It can be seen from the figure and table that when the lowerDropThreshold is quite small, the throughput is reduced significantly due to the large number of RTOs.

From the same figure and table it can be observed that when the lowerDropThreshold is set to 60-80 Kbits, both the throughput and number of RTOs are acceptable. However, it is important to notice that in this situation, the TCP end-to-end delay with bandwidth 36 kbps is considerable, i.e. more than one second. Considering that the rather low bandwidth e.g. tens of kbps should not be often the reality in a LTE system, the lowerDropThreshold is proposed to 60-80 Kbits.

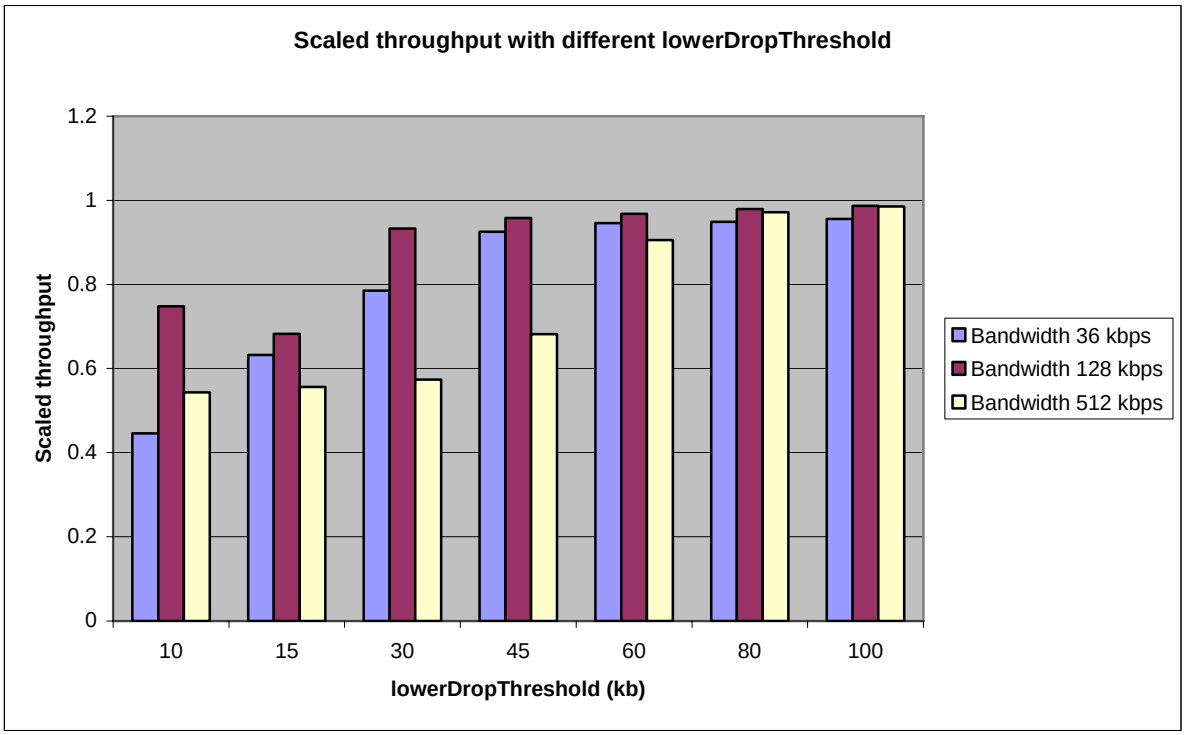

Figure 7.11: *Scaled throughputs with different lowerDropThreshold*

Table 7.4: Number of RTOs with different lowerDropThreshold and bandwidth

lowerDropThreshold		10 Kbits	15 Kbits	30 Kbits	45 Kbits	60 Kbits	80 Kbits	100 Kbits
36 kbps	Number of RTOs	559	213	81	4	4	4	4
	Mean TCP e2e delay (s)	0.63	0.83	1.06	1.20	1.70	2.02	2.50
128 kbps	Number of RTOs	10	52	0	0	0	0	0
	Mean TCP e2e delay (s)	0.28	0.27	0.31	0.37	0.53	0.67	0.85
512 kbps	Number of RTOs	19	20	18	10	1	0	0
	Mean TCP e2e delay (s)	0.16	0.16	0.16	0.16	0.16	0.17	0.19

7.3. Performance comparison of different queue management algorithms

Section 7.2 has investigated the performance of R-AQM and some optimal values have been proposed for each parameter. In this section the performance of R-AQM will be compared to other queue management algorithms i.e. T-AQM, PDCP discard [12] and drop-from-front. The performance mainly focuses on two subjects: the throughput and the TCP end-to-end delay. A good algorithm should maintain both of them in an acceptable level, i.e. a good throughput as well as a low end-to-end delay.

Additionally, it should be noticed that when the bandwidth down-switches, the RLC SDUs in the buffer may get long delay until the TCP adapts to the new PC. Thus, both the mean delay and the maximum delay are interesting and investigated in this section.

7.3.1. Performance comparison with a fixed bandwidth

In this section, a number of simulations are done to investigate the performance of different queue management algorithms in a fixed bandwidth scenario. The corresponding parameters for T-AQM and R-AQM are configured the same, i.e.

	minAgeThreshold	minInterDropTime	lowerDropThreshold	maxAgeThreshold
R-AQM	0.2 s	0.6 s	60000 Kbits	N/A
T-AQM	0.2 s	0.6 s	5 packets	0.8 s

It should be noticed that the lowerDropThreshold of R-AQM and T-AQM are different. The lowerDropThreshold used in [4] denotes the number of packets. However, for R-AQM, the BSR only reports the amount of bits in the buffer but not the number of packets; thus the lowerDropThreshold in R-AQM denotes the bits. Additionally, there is no maxAgeThreshold in R-AQM while in T-AQM it is set to 0.8 s. In most of the cases the maxAgeThreshold will not have any effect, unless the bandwidth down-switched quite a lot and the data is stuck in the buffer. The other settings are:

- File size: 100 Mbytes
- Numbers of file transfers: 1

Figure 7.12 and Table 7.5 illustrate the scaled throughputs and TCP end-to-end delays when transferring one large file for the terminal to the server. Scaled throughput means the real throughput is divided by the optimal throughput, which is the throughput when the buffer is infinite. From the figure it can be seen that the throughputs are quite similar, i.e., higher than 95% of maximum in most situations except for the drop-from-front with low bandwidths. The reason for the throughput degradation is that the drop-from-front algorithm maintains the queue until the buffer is full that may cause the buffer overflow resulting RTOs. In these simulations, when using the drop-from-front algorithm, the numbers of RTOs are 2, 3 and 5 with bandwidths of 640, 384 and 128 kbps, respectively. As a comparison, the numbers of RTOs are 0, 1 and 1 when using R-AQM with the same bandwidths.

On the other hand, if we investigate the TCP end-to-end delay, the delays of drop-from-front in a low bandwidth scenario are up to tens of seconds and they are definitely unacceptable. Other three algorithms have quite similar delays despite the PDCP discard which has a little bit smaller mean and maximum delay than the other two.

Previous simulations are focus on transferring a large i.e. 100 Mbytes file. However, in reality, we usually transfer much smaller files from time to time, e.g. web objects, mp3 files, flash video etc. In these kinds of situations, how the AQM algorithms react with the TCP initial slow-start phase is important. Some simulations are done to investigate the performance when quite a few small files are transferred. These simulations have the same parameters as above except that the file size and numbers of file transfers. Additionally, between the file transfers, there is random idle time.

- Mean file size: 300 Kbytes
- File size distribution: exponential
- Numbers of file transfers: 50
- Mean idle time: 8 s
- Maximum idle time: 10 s
- Minimum idle time: 5 s

Figure 7.13 and Table 7.6 illustrate the performance of these simulations. Comparing to the single file transfer scenario, the drop-from-front has better performance, both end-to-end delay and throughput in a low bandwidth scenario. The reason is the transfers of small files are usually completed before the buffer is full. Hence there are few overflow situations and the end-to-end delay is reduced.

On the other hand, the PDCP discard have worse performance in some bandwidth e.g. 10 Mbps and 3 Mbps. To investigate this problem, the *cwnd* and the RLC buffer are illustrated in Figure 7.14, in which with the bandwidth of 3 Mbps. It can be observed that in the initial slow-start phase, the *cwnd*, which is indicated by the black triangles, increases quite fast; thus the queue size in RLC buffer, which is indicated by blue crosses, accumulates quickly; thus, the Pipe Capacity can not absorb the queue fast enough and multiple packets in the queue exceed the PDCP drop threshold and are discarded, the dropped packets are indicated as orange diamonds in Figure 7.14. The multiple packets drop causes the RTOs and degrade the performance significantly. From Figure 7.14 we can see that from 3rd second to 6th second there is quite little data to be transmitted. Furthermore, it is important to point out that in this simulator, the SACK TCP is implemented; as discussed in Section 3.3.3, the SACK improves the performance when multiple packets drops in one *cwnd*. Thus, we can image that if some older TCP e.g. new Reno is used, the performance may even worse.

As discussed above, we can conclude that in a fixed bandwidth scenario, both the T-AQM and R-AQM have good performance. They maintain the throughput up to 98% comparing to the optimal throughputs. The TCP end-to-end delays, on the other hand, are less than 0.5 s in most of environments. Note that the R-AQM usually has a little larger delay than the T-AQM. The reason is that there is a time interval between each BSR and the estimation of delay is discontinuous, thus the R-AQM usually does not respond as fast as T-AQM. The PDCP discard also performs well in most situations. However, it has problem to handle the TCP initial slow-start phase, in which multiple packets are discarded. It may cause the RTOs and degrade the throughput, especially when transferring multiple small files; e.g. the throughput is only around 70% in a scenario of 3 Mbps bandwidth in Figure 7.13. The drop-from-front approach has severe problem when the bandwidth is low: The buffer overflow causes throughput degradation. On the other hand, the end-to-end delays are huge due to the large buffer.

Figure 7.12: *Scaled throughputs when transferring one large file*

Table 7.5: The mean and maximum TCP end-to-end delay with different bandwidths when transferring a large file

		R-AQM	T-AQM	PDCP discard	Drop-from-front
50 Mbps	Mean delay (s)	0.1557	0.1328	0.1332	0.1328
	Max delay (s)	0.341	0.263	0.239	0.263
30 Mbps	Mean delay (s)	0.1241	0.0983	0.0967	0.2059
	Max delay (s)	0.377	0.367	0.247	0.408
10 Mbps	Mean delay (s)	0.14	0.1328	0.1316	0.5881
	Max delay (s)	0.38	0.352	0.243	1.128
3 Mbps	Mean delay (s)	0.1721	0.1655	0.1649	1.9682
	Max delay (s)	0.386	0.349	0.258	3.791
1 Mbps	Mean delay (s)	0.1797	0.1779	0.1762	1.891
	Max delay (s)	0.373	0.366	0.269	11.45
640 kbps	Mean delay (s)	0.1875	0.1843	0.1838	8.6116
	Max delay (s)	0.383	0.368	0.276	13.886
384 kbps	Mean delay (s)	0.1971	0.1955	0.1934	11.984
	Max delay (s)	0.43	0.4	0.287	23.769
128 kbps	Mean delay (s)	0.5393	0.5368	0.2753	19.632
	Max delay (s)	0.886	0.813	0.449	82.827

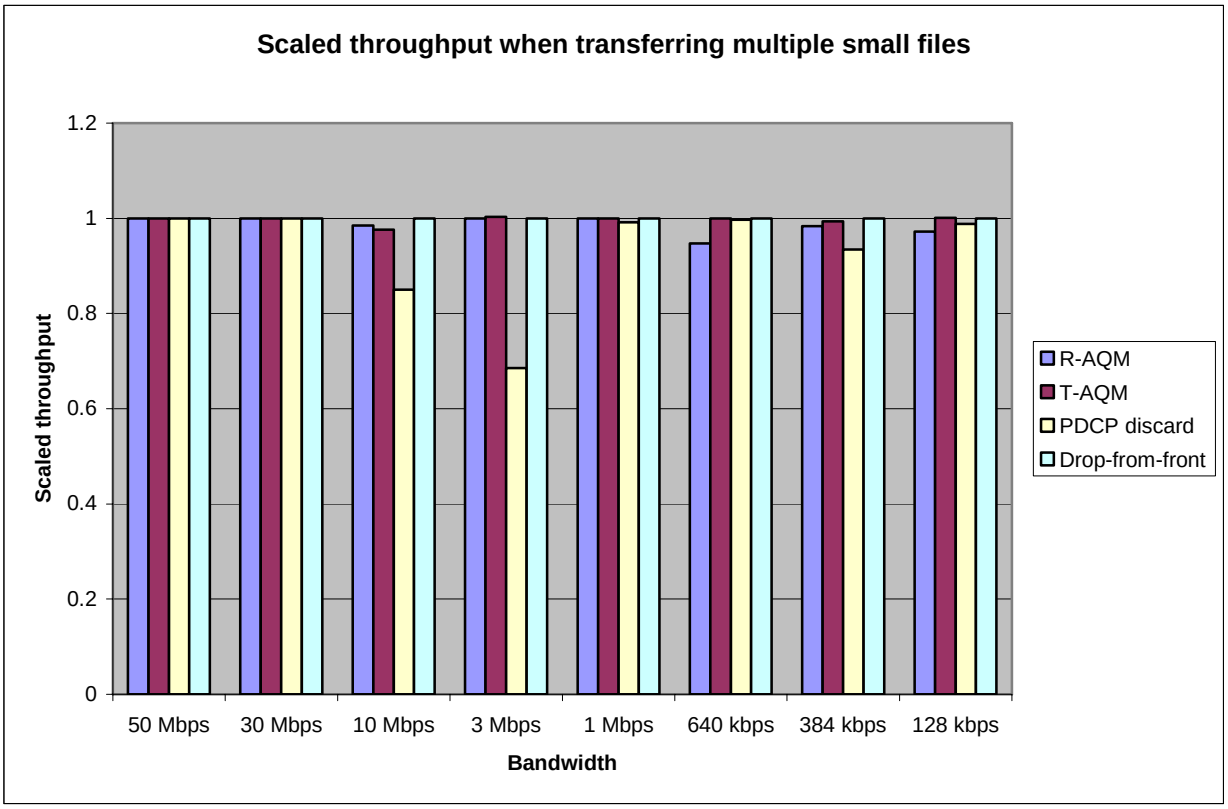

Figure 7.13: *Scaled throughputs when transferring multiple small files*

Table 7.6: The mean and maximum TCP end-to-end delay with different bandwidths when transferring multiple small files

		R-AQM	T-AQM	PDCP discard	Drop-from-front
50 Mbps	Mean delay (s)	0.08	0.08	0.08	0.08
	Max delay (s)	0.141	0.141	0.141	0.141
30 Mbps	Mean delay (s)	0.091	0.091	0.091	0.091
	Max delay (s)	0.206	0.206	0.206	0.206
10 Mbps	Mean delay (s)	0.13	0.128	0.112	0.138
	Max delay (s)	0.426	0.373	0.247	0.613
3 Mbps	Mean delay (s)	0.186	0.178	0.124	0.426
	Max delay (s)	0.445	0.408	0.264	2.192
1 Mbps	Mean delay (s)	0.192	0.188	0.178	1.333
	Max delay (s)	0.434	0.416	0.297	6.82
640 kbps	Mean delay (s)	0.1626	0.1971	0.1846	1.1518
	Max delay (s)	0.383	0.418	0.287	4.294
384 kbps	Mean delay (s)	0.1943	0.2066	0.1952	1.9962
	Max delay (s)	0.43	0.434	0.295	7.409
128 kbps	Mean delay (s)	0.5317	0.5335	0.2804	6.9764
	Max delay (s)	0.867	0.758	0.452	25.671

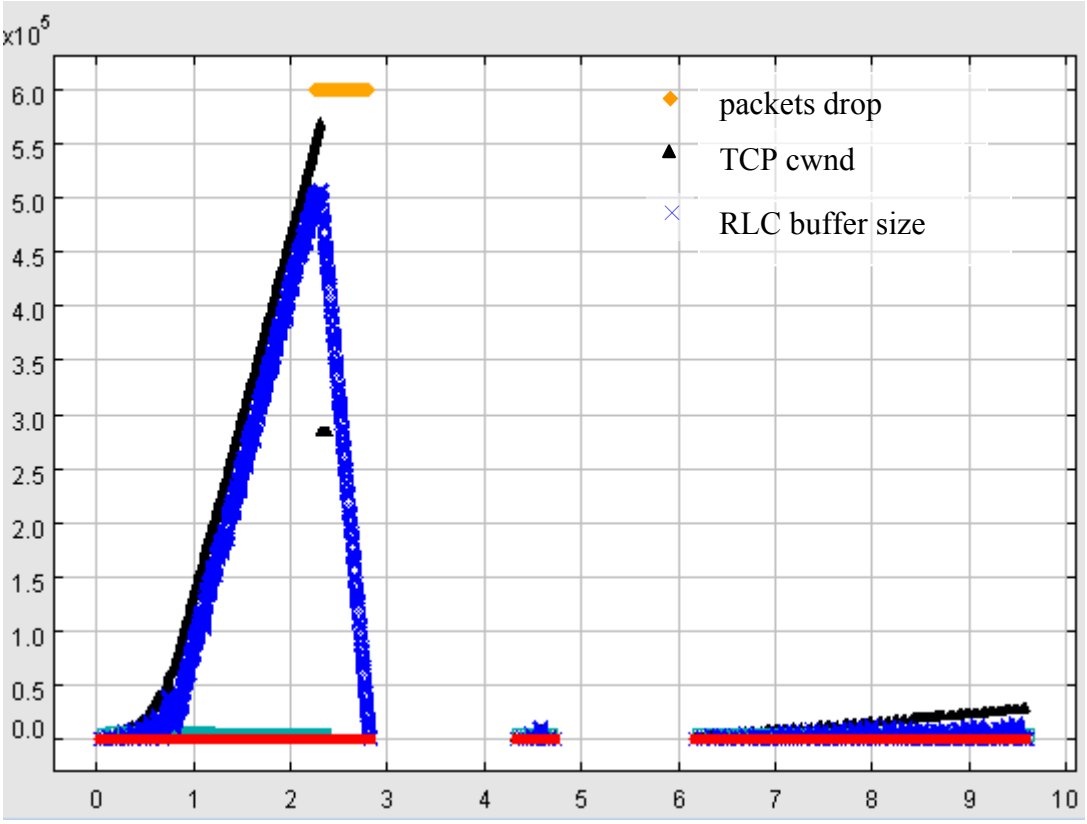

Figure 7.14: *Multiple packets drop with PDCP discard algorithm in 3 Mbps bandwidth*

7.3.2. Performance comparison in varying bandwidth

In Section 7.3.1 the performance of different algorithms are compared in a fixed bandwidth scenario. However, in mobile communication, one major characteristic is bandwidth variation. In this section, the performances of different algorithms are compared in a scenario with deterministically varying bandwidth; that is, the bandwidth oscillates between two predefined values. Note that the oscillation comprises four categories: fast and low varying, fast and high varying, slow and low varying, slow and high varying. It is also presented in [4] that when the bandwidth oscillates very fast e.g. less than 1s, it is actually averaged from the TCP's point of view. Thus, the "fast" here means the period of oscillation is the order of 1-5s while "slow" is 10-20s.

As discussed above, the bandwidths of simulations in this section are

	Low bandwidth (Mbps)	High bandwidth (Mbps)	Period (s)
Fast and low	3	10	2
Fast and high	0.128	2	3
Slow and low	0.512	1.5	15
Slow and high	2	20	20

Note that the bandwidth oscillation includes 2 different phases which are bandwidth up-switch and down-switch. As discussed in [4], when a large down-switch happens, the queue is heavily over-dimensioned and the TCP end-to-end delay may become huge. In T-AQM, the maxAgeThreshold is used to drop consecutive packets to drain the queue quickly. However, in R-AQM, it is difficult to drain the buffer fast due to the property of discarding the packets at receiver side. Nevertheless, there is a possible solution: combining the R-AQM and PDCP discard. That is to say, the PDCP discard is implemented in UE side while the R-AQM is implemented in eNodeB side. Additionally, the drop threshold of PDCP discard should be configured large enough, e.g. 0.8 s-1 s. In such an implementation, only the R-AQM takes effect in most situations; on the other hand, in some special situation, e.g. large bandwidth down-switch, the PDCP discard can take effect to drain the buffer fast, just as the maxAgeThreshold does with T-AQM. In this section, the performance of combination with R-AQM and PDCP discard will also be illustrated; it is called combined-AQM in the following text. However, it should be pointed out again that the PDCP discard is actually implemented in RLC buffer.

Figure 7.15, Figure 7.16 and Figure 7.17 illustrate the RLC buffer status using T-AQM, R-AQM and combined-AQM respectively when the bandwidth switches from 10 Mbps to 500 kbps at 20 second. It can be seen that the T-AQM has drained the buffer only after 1 s of the bandwidth down-switch; on the other hand, the R-AQM spends 5s to absorb the whole queue in the buffer. As a result, the end-to-end delay of R-AQM may inflate to large. The combine-AQM performs quite similar to the T-AQM. Additionally, it is worthy to point out that in Figure 7.17 the RLC buffer size changed to zero and then suddenly to 500 Kbyte at the time 19s. However, there is no reasonable explain about that and we assume it is a bug of the simulator.

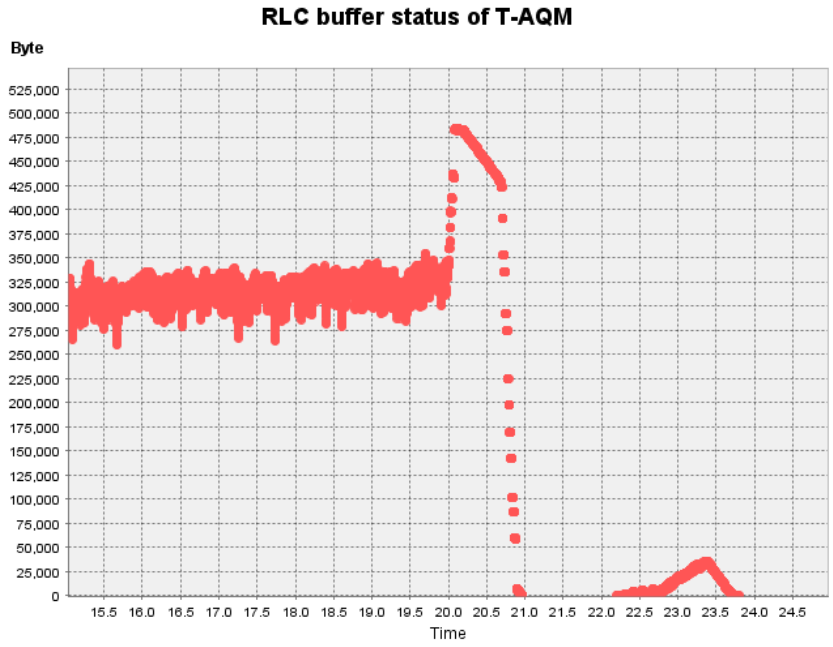

Figure 7.15: *Buffer status of T-AQM when bandwidth down-switch*

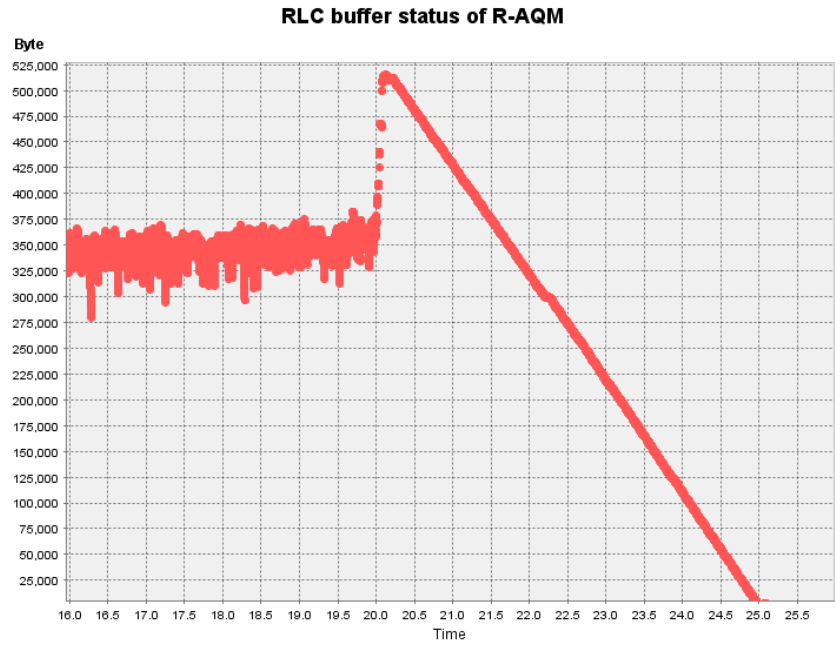

Figure 7.16: *Buffer status of R-AQM when bandwidth down-switch*

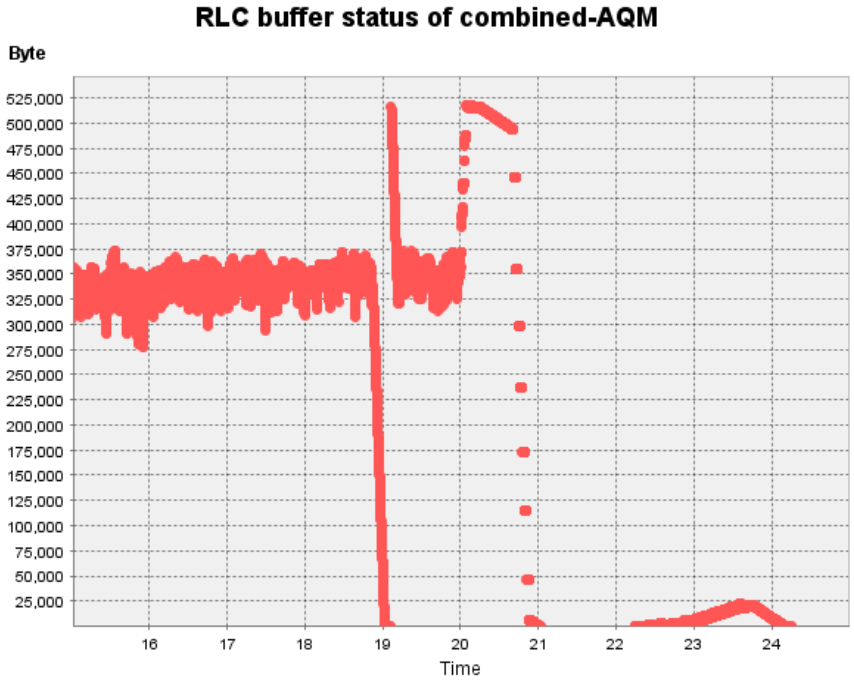

Figure 7.17: *Buffer status of combined-AQM when bandwidth down-switch*

On the other hand, when the bandwidth up-switch happens, it may cause the link under-utilization. The reason is that the queue size in a low bandwidth scenario is quite small and not enough to feed the inflated Pipe Capacity. Thus, in a bandwidth oscillation environment, the delay and throughput may differ from the case with the fixed bandwidth.

The general parameters of these simulations are

- File size: 10 Mbytes
- Numbers of file transfers: 10

Figure 7.18 illustrates the throughput of different queue management algorithms with different bandwidths. From the figure we can see that in most situations, the drop-from-front has the best throughput comparing to the other three delay-based algorithms. The reason is that the delay-based algorithms will suffer from link under-utilization when the bandwidth up-switches, as discussed above. Furthermore, when there is a large bandwidth down-switch, the queue is stuck in the buffer. Thus the packets in this queue will suffer large delay and exceeds the maxAgeThreshold of T-AQM as well as the drop threshold of PDCP discard, resulting multiple consecutive drops and reducing the throughput significantly.

It is worthy to point out that when the bandwidth oscillates between 2 Mbps and 20 Mbps with period of 20 s, the throughput of R-AQM and PDCP discard seems to have an obvious degradation comparing to T-AQM. In fact, the throughput degradation is not that large. The reason is that the T-AQM completed the file transferring around 220 s and from 200 s to 220 s is the bandwidth of 20 Mbps. While the R-AQM and PDCP discard completed at 226 s and from 220 s to 240s is the lower bandwidth 2 Mbps. Thus, if we calculate the throughput before 220 s, the degradation of R-AQM and PDCP discard is only a few percentages.

Though the throughput of drop-from-front is the best, it has severe problem of the TCP end-to-end delay. Figure 7.19 and Figure 7.20 show the mean delay and maximum delay of different algorithms, respectively. From Figure 7.19 we can see that the mean delay of drop-from-front is much larger than that of other three approaches, e.g. when the bandwidth oscillates between 521 kbps and 1.5 Mbps, the mean delay of drop-from-front is up to 1.8 s while the other three are under 0.2 s. The R-AQM, on the other hand, also has problem of large maximum delay when the bandwidth changes significantly. When the bandwidth oscillates between 0.128 Mbps and 2Mbps, the maximum delay is up to 2 s. It should be noted that in realistic radio network, the bandwidth variation may even larger.

The PDCP discard, T-AQM and combined-AQM, on the other hand, maintain good end-to-end delay, however, there is obvious throughput reduction comparing to drop-from-front. This is a drawback of delay-based AQM algorithm: when the bandwidth varies significantly, the throughput may suffer considerably.

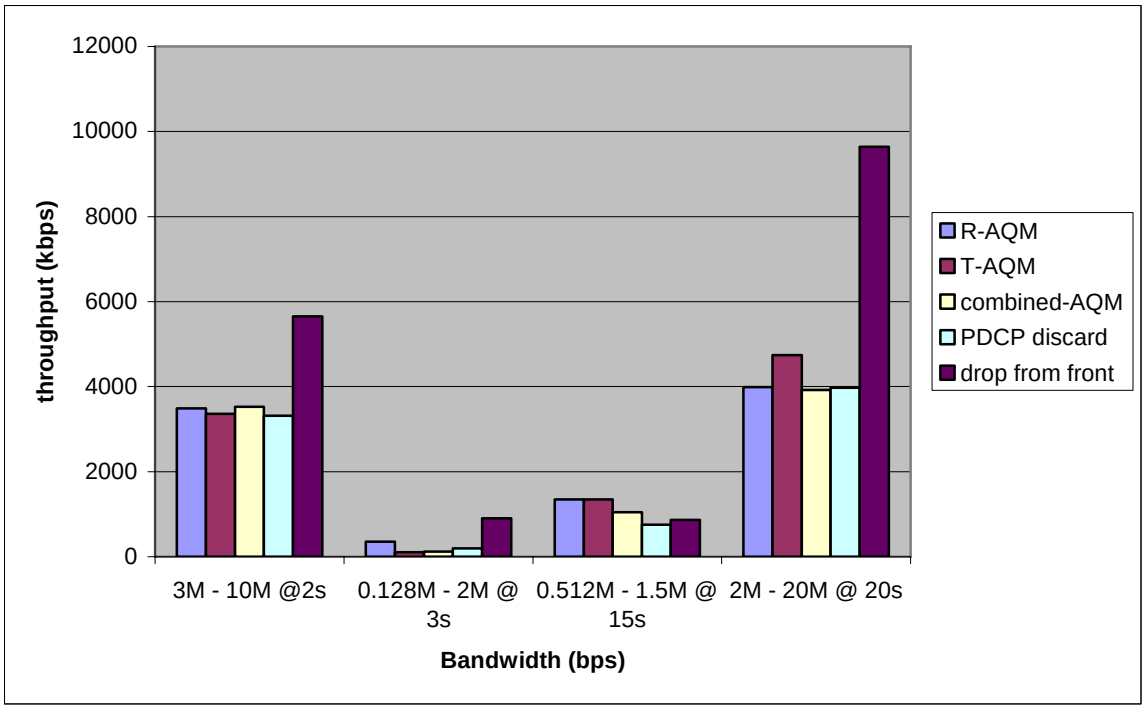

Figure 7.18: *Throughput with varying bandwidths*

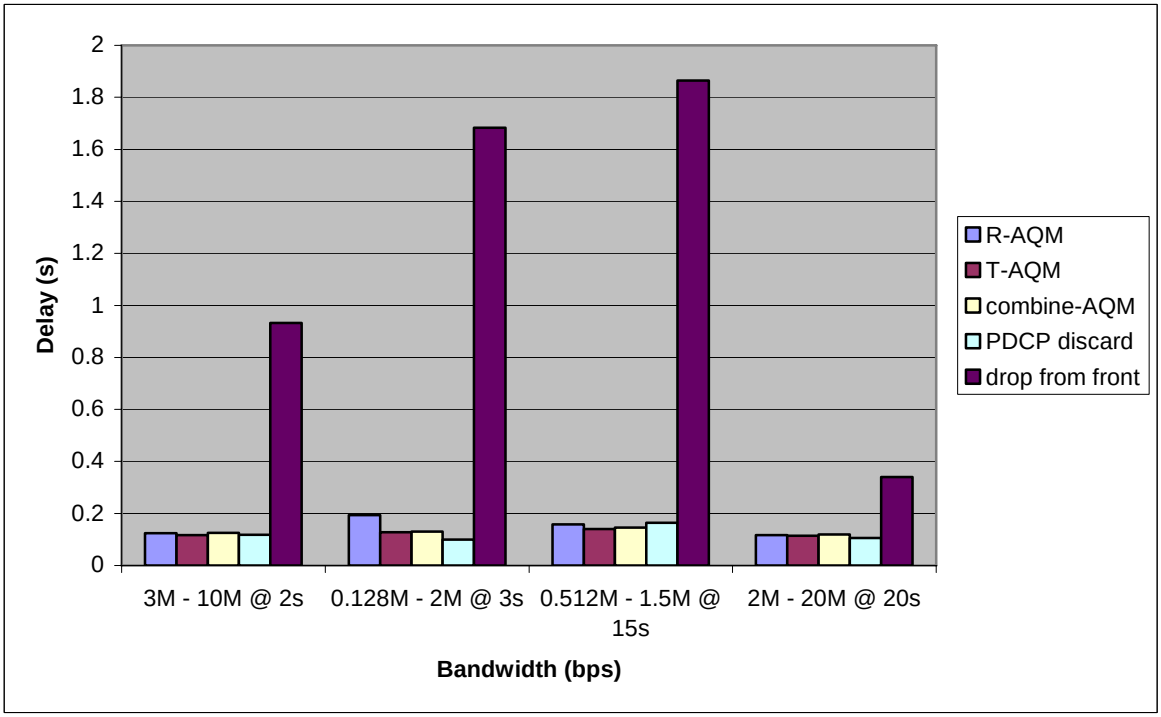

Figure 7.19: *Mean end-to-end delay with varying bandwidths*

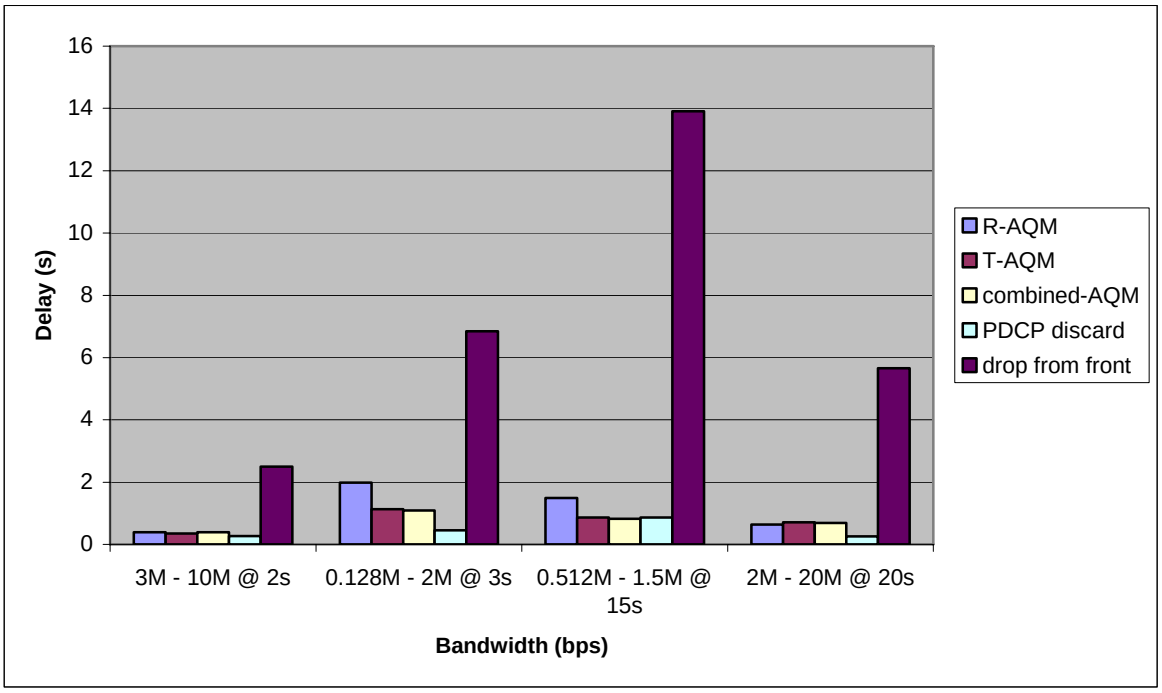

Figure 7.20: *Maximum delay with varying bandwidths*

7.3.3. Performance comparison in realistic radio network scenario

Section 7.3.1 and 7.3.2 compare the performance of different queue management algorithms in deterministic bandwidth scenarios. However, in a realistic radio network, the bandwidth varies quite randomly, depending on the instantaneous channel quality as well as number of active users in the cell. In this section, some simulations are done to investigate the performance of different queue management algorithms in realistic radio network.

Furthermore, two environments are investigated in this section, the terminals have different speed, one has 0.83 m/s, which is assumed to be in a pedestrian and the other has 30 m/s, which is assumed to be in a car. The throughput and end-to-end delay are illustrated in Figure 5.21 and 5.22, where the corresponding speed are 0.83 and 30 m/s, respectively.

From the Figures it can be observed that when the speed is low, the average bit rate is high and vice versa. Furthermore, the R-AQM is as good as the T-AQM, both as regard to throughput and end-to-end delay. The drop-from-front has best throughput but with a larger delay than the others. PDCP discard has worse throughput because it triggered two much RTOs, as Table 7.7 shows. However, it should be noticed again that the Drop-from-front may have worse throughput if some other TCP protocol is implement than SACK. The buffer overflow actually happened as shown in Figure 7.23, which depicts the CDF of queue size. As comparison, the other algorithms only used quite smaller buffer.

Table 7.7: Time of RTOs in a realistic radio network

	R-AQM	T-AQM	PDCP discard	Drop-from-front
Terminal speed 0.83 m/s	0	0	5	0
Terminal speed 30 m/s	1	0	10	0

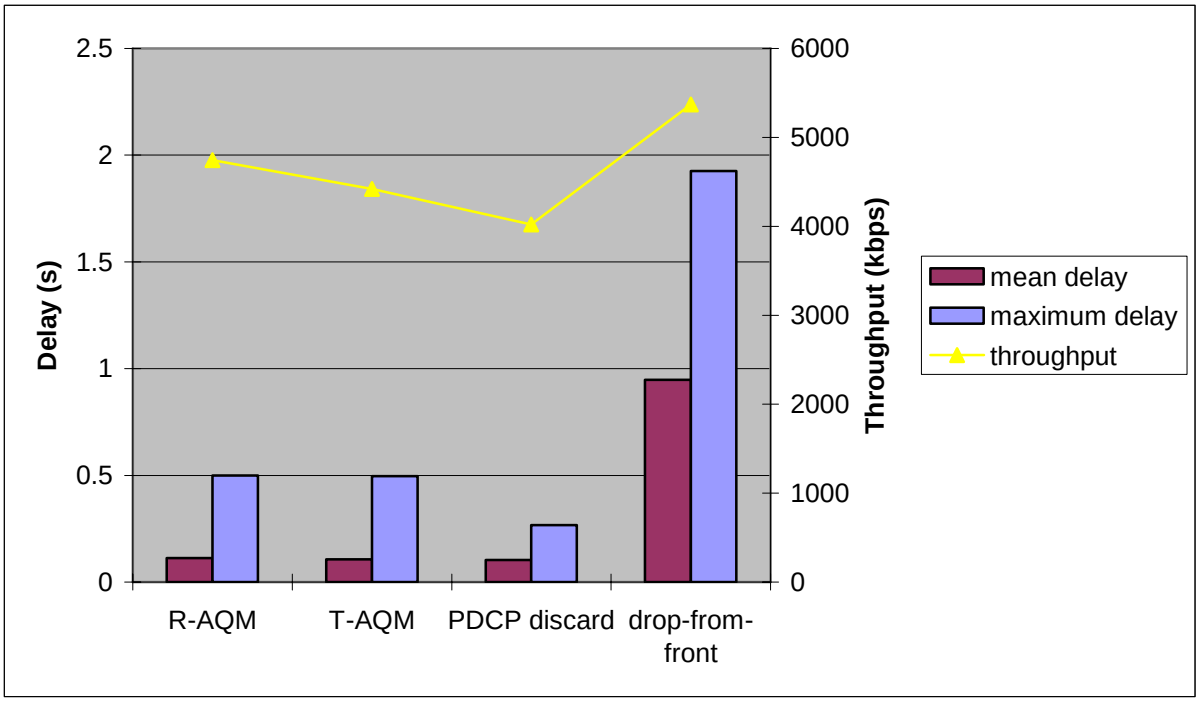

Figure 7.21: *Throughput and end-to-end delay in a realistic radio network with UE speed 0.83 m/s*

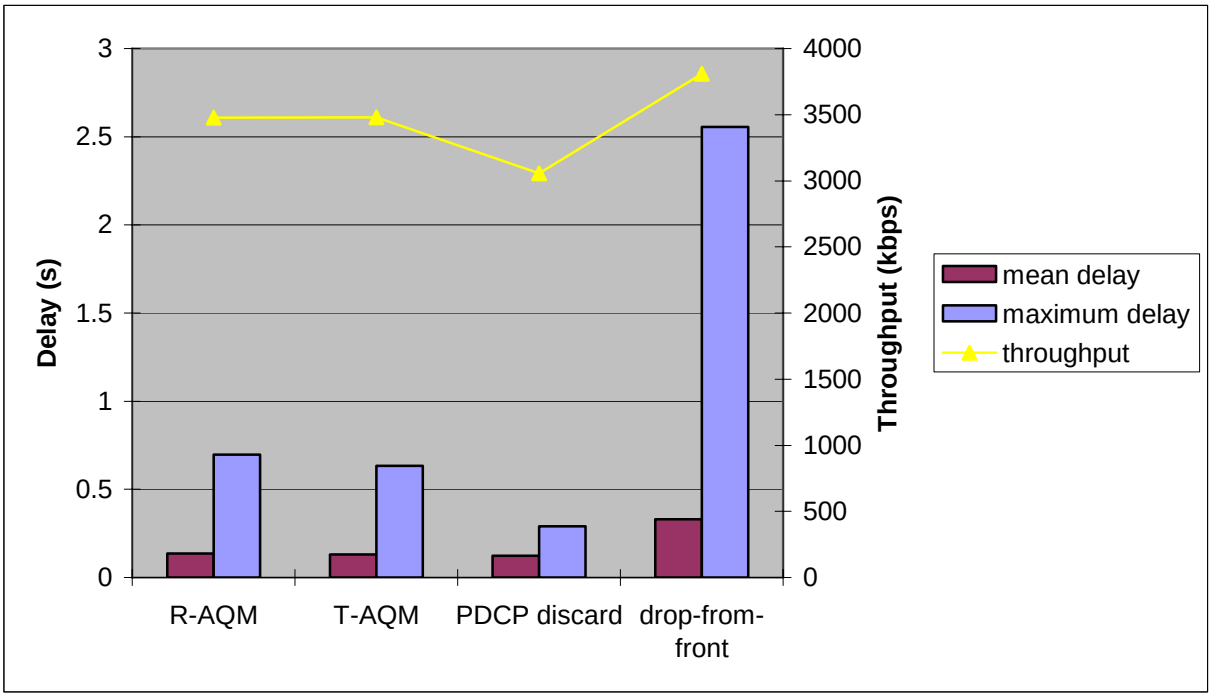

Figure 7.22: *Throughput and end-to-end delay in a realistic radio network with UE speed 30 m/s*

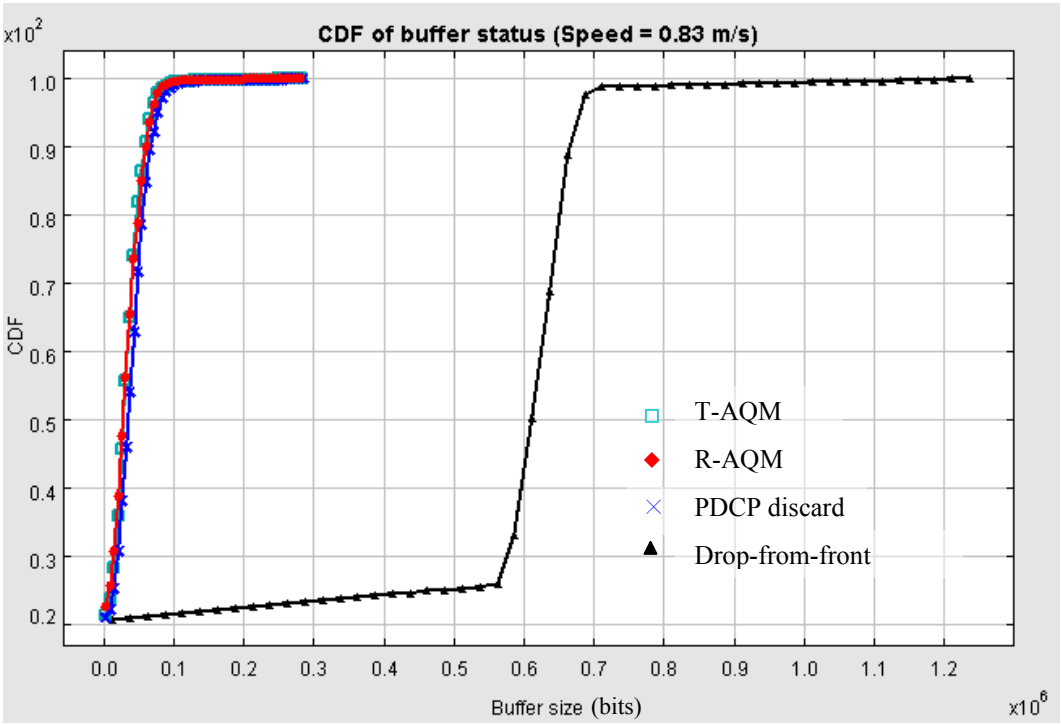

Figure 7.23: *CDF of buffer status in realistic radio network with UE speed 0.83 m/s*

7.3.4. Performance comparison with multiple flows

As discussed in Chapter 4, one problem of passive queue management is unfair sharing, which means one or a few TCP flows dominate the buffer and lock out other new coming TCP flows. In [4] it has been proved that the T-AQM improves the fairness of bandwidth sharing considerable. This section will compare the fairness issues of R-AQM with other different queue management algorithms.

In these simulations, one huge TCP flow is firstly dominating the bandwidth and the buffer; then some small files come randomly and try to share the bandwidth. The throughputs of these small files will be investigated to see how largely the flows actually share the bandwidth. The general parameters are

- Flow 1 file size: 300 Mbytes
- Numbers of file transfers: 1
- Flow 2 file size mean value: 300 Kbytes
- Size distribution: Exponential
- Number of files: 10

Figure 7.24 shows the average throughput of the small files. It is obvious that the drop-from-front has severe problem of unfair sharing. When the bandwidth is high, e.g. 20 Mbps and 3 Mbps, the drop-from-front locks out the new TCP flows. When the bandwidth is quite low, e.g. 384 kbps, the throughput of drop-from-front is acceptable but the end-to-end delay is large, similar to previous sections.

On the other hand, the T-AQM, R-AQM and PDCP discard have quite similar performance. However, when the bandwidth is quite high, e.g. up to 20 Mbps, the throughput of PDCP discard is the best. The reason is that the R-AQM and T-AQM define the parameter minInterDropTime to prevent consecutive drop. The value is set to 0.6 s in these simulations. The new coming TCP flow starts with an aggressive slow-start phase and very soon some packets will exceed the minAgeThreshold. Thus a drop is triggered. Since the drop is random, the discarded packet may belong to the new comer or the dominating one. If it belongs to the new comer, then the drop will cause the new TCP flow to halve its *cwnd* and wait another 0.6 s to trigger the next drop. During this time, the old one still dominates the bandwidth. For PDCP discard, there is no requirement for waiting some time between two drops, thus the dominating one usually have packet drop earlier than T-AQM and R-AQM. Thus the PDCP discard usually maintain the best fairness of bandwidth sharing. However, the consecutive drops may cause the whole link under-utilization. Consequently, even the PDCP discard maintain better fairness than T-AQM and R-AQM, the throughput may be worse, like Figure 7.14 shows in a case when the bandwidth is 3 Mbps.

As discussed above, the PDCP discard maintains the best fairness issue. However, R-AQM and T-AQM perform more balanced functionality in maintaining both good fairness and full link utilization. The drop-from-front, on the other hand, has severe problem on fair sharing bandwidth.

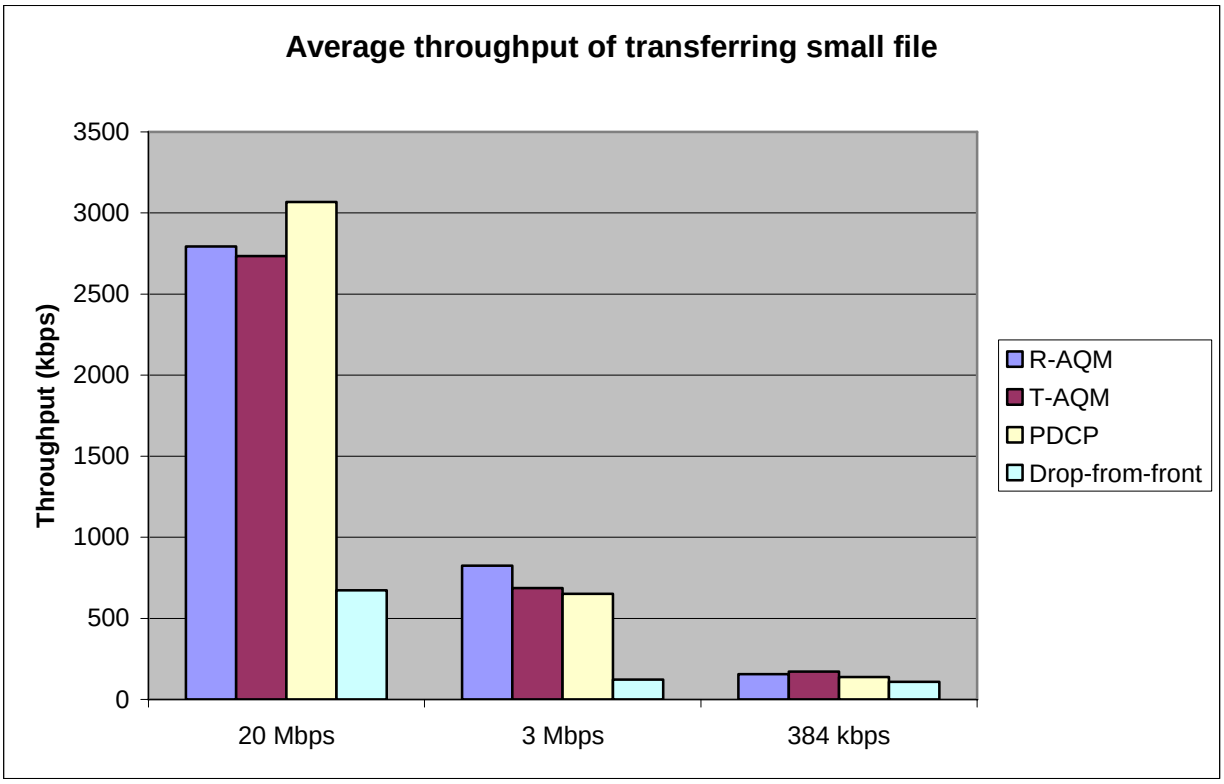

Figure 7.24: *Average throughput of other small files when one TCP is dominating the bandwidth*

7.3.5. The impact of BSR period

The R-AQM estimates the packet delay based on the Buffer Status Report (BSR). Therefore, the time interval of each BSR has impact on the performance of R-AQM. Furthermore, the BSR also influences the uplink scheduling decision of eNodeB; thus it may has impact on the performance of T-AQM and PDCP discard as well, since the uplink scheduling controls the data transmission from the UE to the eNodeB. This section will investigate how the BSR period affects the performance of different queue management algorithms.

Figure 7.25 and Figure 7.26 illustrate the throughput with different BSR periods. When the period is quite small, e.g. 10 ms or 20 ms, the throughput of the four algorithms are quite similar. However, when the BSR period increases, those delay-based algorithms i.e. T-AQM, R-AQM and PDCP discard suffer throughput degradation. For example, when the BSR is 100 ms, the throughput of T-AQM and PDCP discard has around 20% throughput reduction in a 20 Mbps bandwidth scenario. Nevertheless, the R-AQM can still maintain the throughput in an acceptable level, which is less than 10%. On the other hand, the drop-from-front has no obvious influence on throughput. This is because the BSR period has impact on scheduling decision of eNodeB. When the period is low, which means more BSR will be transmitted and the eNodeB can schedule the uplink transmission more frequently. For delay-based AQM, this means the packets can leave the buffer more quickly thus few packets will be discarded. On the other hand, if the period is large, the eNodeB will schedule the uplink transmission with long time intervals. As a consequence, the packets may wait longer and more packets may exceed the drop threshold and be discarded. The R-AQM, however, is more robust against on the BSR period

since it estimates the delay according to received BSRs; less BSRs means less estimation and results less packet drops. The cost is a larger end-to-end delay, as seen from Table 7.8 and Table 7.9. Meanwhile, when using a queue-size-based algorithm e.g. drop-from-front, the delays of the packets are not taken into account when dropping packets. Thus, this kind of algorithm is not influenced by the BSR period length.

As discussed above, the BSR period clearly has impact on the performance, especially for the delay-based AQM algorithms. However, from the figures we can see that the R-AQM is more robust against the BSR period inflation than T-AQM and PDCP discard, with the cost of enlarged end-to-end delay. On the other hand, the Drop-from-front is not influenced by the BSR period.

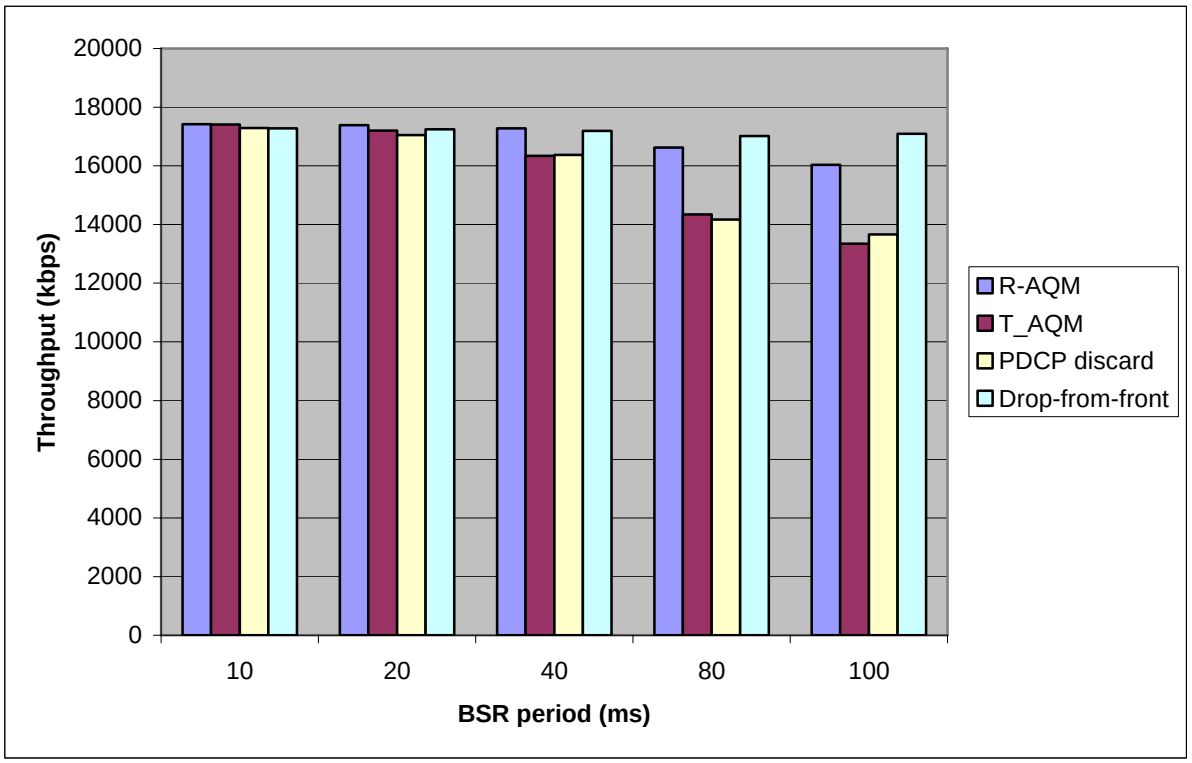

Figure 7.25: *Throughput with different BSR period in 20 Mbps bandwidth*

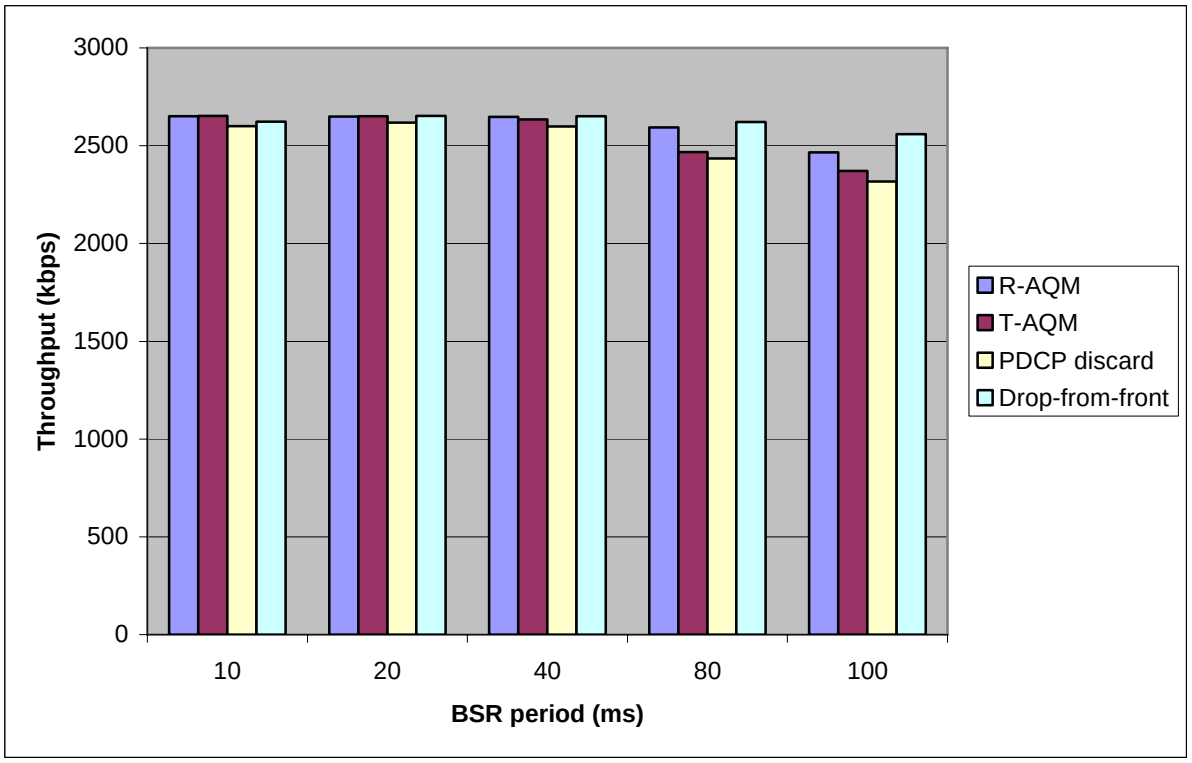

Figure 7.26: *Throughput with different BSR period in 3 Mbps bandwidth*

Table 7.8: Mean delay with different BSR period in 20 Mbps bandwidth

	R-AQM	T-AQM	PDCP discard	Drop-from-front
10ms	0.1243	0.1157	0.114	0.6001
20ms	0.1263	0.1171	0.1158	0.601
40ms	0.1465	0.1245	0.122	0.5951
80ms	0.2101	0.1474	0.1462	0.592
100ms	0.1746	0.1571	0.1581	0.5921

Table 7.9: Mean delay with different BSR period in 3 Mbps bandwidth

	R-AQM	T-AQM	PDCP discard	Drop-from-front
10ms	0.1714	0.1657	0.1644	3.789
20ms	0.1721	0.1655	0.1649	3.791
40ms	0.1762	0.1661	0.1653	3.791
80ms	0.1997	0.1763	0.1756	3.7821
100ms	0.1938	0.1834	0.1828	2.538

5.3.6 Performance if both R-AQM and T-AQM are implemented

As presented in previous sections, the R-AQM can perform quite similar functionalities as T-AQM. However, there is still a potential problem: What is the performance when the both algorithms are used at the same time, i.e., an UE having T-AQM implemented is served by an eNodeB with R-AQM implemented. To investigate this problem, some simulations are done again when both the T-AQM and R-AQM implemented.

Table 7.10 shows the throughput in fixed bandwidth, with the same parameter as in Section 7.3.1. As comparison, the throughputs of R-AQM and T-AQM are also provided. From the table it can be observed that the throughputs are quite similar to each other. On the other hand, the mean and maximum TCP end-to-end delay is illustrated in Figure 7.27 and Figure 7.28.

As discussed above, it can be concluded that when a UE with T-AQM served by an eNodeB with R-AQM, the performance will not suffer obvious degradation.

Table 7.10: Throughput if both AQM are implemented

	30 Mbps	3 Mbps	384 kbps
R-AQM	25358	2649.6	316.44
T-AQM	25367	2650.4	320.84
Both AQM	25367	2649.8	316.77

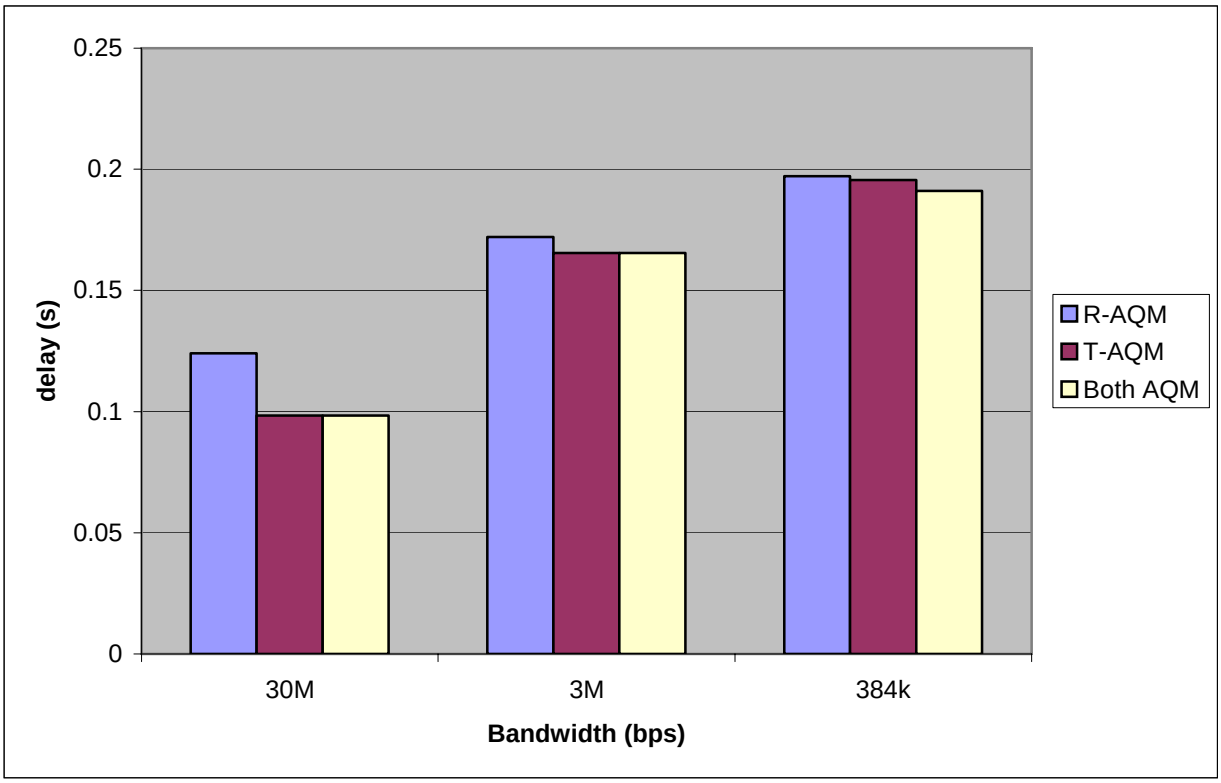

Figure 7.27: *Mean TCP end-to-end delay in different bandwidth*

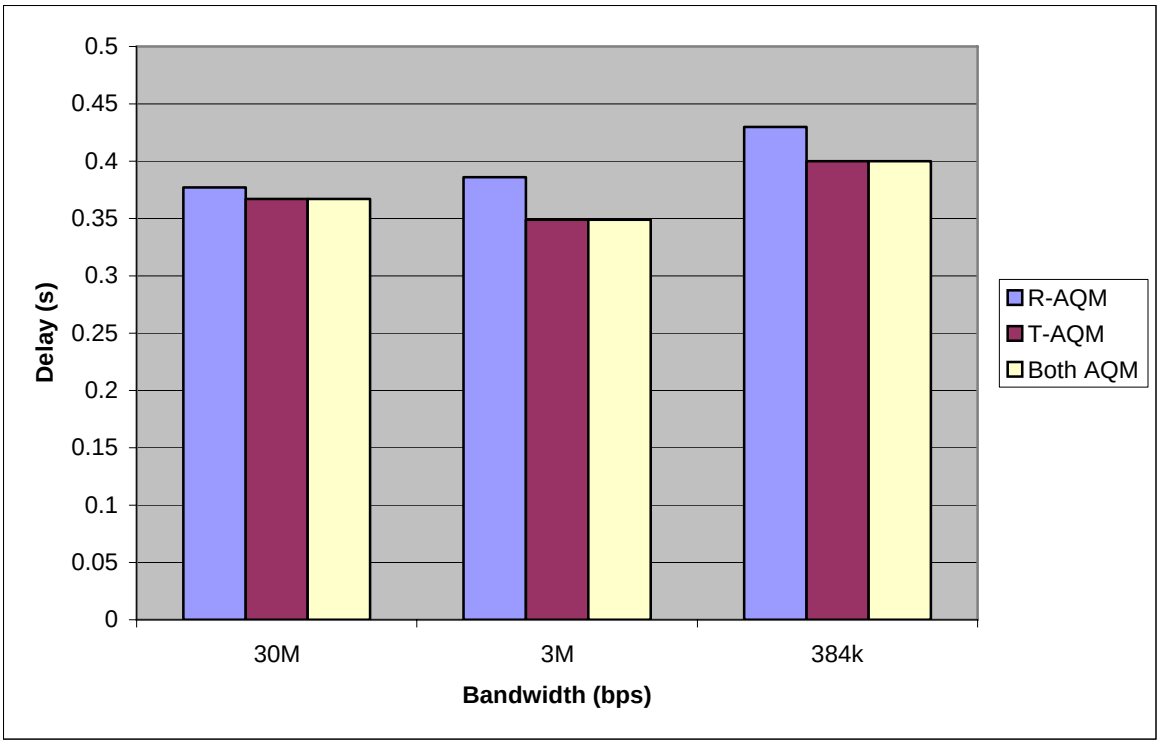

Figure 7.28: *Mean TCP end-to-end delay in different bandwidth*

7.4. Conclusions from simulation results

In this section, a summary of simulation results will be done. First, Section 7.2 proposes some optimal values for each parameter of R-AQM based on simulations. The minAgeThreshold is proposed to be set around 0.2 s. However, it is important to point out that the simulations are done with 40ms Internet delay. Generally speaking, a larger Internet delay requires a larger minAgeThreshold. The lowerDropThreshold, on the other hand, is proposed to be set 60-80 Kbits. This value can improve the throughput in low bandwidth cases, when the minAgeThreshold is too small to maintain full link-utilization. Finally, the optimal value of minInterDropTime is above 0.6 s. It can prevent consecutive drops thus efficiently reduce the number of RTOs.

In Section 7.3 numerous simulations are done to compare the performance of four queue management algorithms, i.e. R-AQM, T-AQM, PDCP discard and Drop-from-front in different environments. The results show that in most situations, the major problem of PDCP discard is lack of protection from consecutive drops. In initial TCP slow-start phase, due to the aggressive increment of TCP *cwnd*, it is easy to cause consecutive drops and degrade the throughput significantly, especially when transferring small files. The drop-from-front algorithm, on the other hand, mainly suffers the over-buffered problem, e.g. the end-to-end delay is quite huge in a low bandwidth case. Additionally, the buffer overflow may cause throughput reduction in some situation.

The T-AQM and R-AQM have similar and satisfying performance; maintain both good throughput as well as low end-to-end delay in most simulations done in Section 7.3. Nevertheless, problems may

arise when the bandwidth varies a lot. Both of T-AQM and R-AQM suffer throughput reduction in such environments. Furthermore, the R-AQM may have a large end-to-end delay during a short period. The reason for the difference is that, when the bandwidth down-switches significantly, the T-AQM can drain the buffer fast thus having a shorter end-to-end delay. Fast draining is handled by defining a parameter called maxAgeThreshold when this threshold is exceeded, the packet will be discarded without considering the minInterDropTime. Due to the property that the eNodeB cannot operate UE's buffer directly, there is no such parameter in R-AQM. An optional solution is combine the PDCP discard with R-AQM. That is to say, in the UE side, the PDCP discard is implemented while in the eNodeB side, the R-AQM is implemented. Furthermore, the drop threshold of PDCP discard should be larger than minAgeThreshold, e.g. around 0.8 s – 1 s. The performance of the combined algorithms is illustrated in Section 7.3.2 and it can be seen that the combination efficiently drains the buffer thus reduces end-to-end delay.

The performance of different algorithm in multiple flows scenario is also investigated by simulations. From the results it can be seen that the PDCP discard provides the fairest sharing of the bandwidth. However, this is usually achieved by consecutive drops, which may cause significant throughput reduction. On the other hand, the T-AQM and R-AQM maintain relatively fair sharing of resources as well as high throughput. Finally, the drop-from-front has severe problem of fairness.

Because the R-AQM is based on the estimated delay which is calculated according to the BSRs, it is necessary to investigate how the BSR period length influences the performance of different algorithms. The simulation results show that the drop-from-front is seldom influenced by the BSR period length. The PDCP discard and T-AQM suffer 10%-20% throughput reduction when the period changes from 10ms to 100ms. The R-AQM also has reduced throughput, but better than T-AQM and PDCP discard.

Finally, a special situation is considered: the performance when both the R-AQM and the T-AQM are implemented. The simulation results show that there is not a big difference in performance between using both AQM algorithms or in using only one of those. Thus, if the R-AQM is implemented in the eNodeB, it will not degrade the performance of those UEs with having the T-AQM implemented concurrently.

8. Conclusions

This thesis develops an Active Queue Management algorithm for LTE uplink to enhance the performance of TCP traffic. As difference to earlier works, the proposed algorithm is implemented in the eNodeB instead of the mobile terminal thus leaving a better control of the method to the network. Since the delay-based algorithms are shown to have advantages over the queue size based algorithms [4], also the proposed algorithm inherits the idea that the packets are discarded based on the queuing delay. Comparing to queue-size-based criteria e.g. PDPC, the delay-based algorithms maintain both throughput and end-to-end delay in an acceptable level, even if the bandwidth varies from tens of kbps to tens of Mbps.

Because the new developed algorithm is supposed to be implemented in eNodeB, the queuing delays of received packets cannot be known directly. Thus, a method to estimate the delay is proposed. The method utilizes the information of received Buffer Status Reports as well as the amount of data delivered from the RLC layer to higher layers in eNodeB. The performance of the delay estimation method is investigated by simulations. The results show that even though the accuracy of estimation is slightly influenced by the HARQ error probability, the estimation performs good enough to be used to indicate the real queuing delay.

Because the delay can be estimated in eNodeB, as well as the queue length is indicated by BSR, the implementation of Receiver-AQM (R-AQM) in the eNodeB is quite straightforward. Similar to Transmitted-AQM (T-AQM) studied in [4], the R-AQM also defines several parameters called minAgeThreshold, lowerDropThreshold and minInterDropTime. Each time the eNodeB receives a BSR element, it firstly compares the BSR reported queue size with lowerDropThreshold; if the BSR reported size is smaller than lowerDropThreshold, no drop is allowed. Otherwise, the delay will be estimated by the method mentioned above. If the estimated delay is larger than minAgeThreshold, a packet will be discarded unless the time interval between the current time and the previous packet drop time is less than minInterDropTime.

The simulation results show that the AQM algorithms T-AQM and R-AQM indeed perform advanced functionalities comparing to the drop-from-front and PDCP discard. They maintain balanced throughput and end-to-end delay, i.e. a higher throughput than PDCP discard and a smaller end-to-end delay than the drop-from-front approach in most situations. Furthermore, the R-AQM can perform quite similar functionalities as T-AQM in most of the scenarios. Only in a scenario where the bandwidth varies dramatically, the R-AQM may have a larger end-to-end delay than the T-AQM. However, this problem can be overcome by combining the PDCP discard and R-AQM, which a simple and standardized approach. Thus, we can conclude that instead of implementing T-AQM in each UE, we have another option to improve the TCP performance in the LTE uplink: implementing R-AQM in the eNodeB.

References

[1] Dahlman, E. Parkvall, S. Sköld, J. Beming, P. *3G Evolution: HSPA and LTE for Mobile Broadband*. Elsevier, 2007.

[2] Jacobson, V. Karels, M.J. Congestion Avoidance and Control. *In Proceedings of SIGCOMM '88*, ACM Press, 1988.

[3] Floyd, S. Jacobson, V. Random Early Detection Gateways for Congestion Avoidance, *ACM/IEEE Transactions on Networking*. August 1993.

[4] Rathonyi, T. Nilsson, M. An Active Queue Management Algorithm for a Dedicated Wireless Uplink Broadband Channel. *Master Thesis*. Lund, September 2007.

[5] Feng, W. Shin, K. Kandlur, D. Saha, D. The BLUE Active Queue Management Algorithms. *ACM/IEEE Transactions on Networking*. August 2002.

[6] Jacobson. V. Modified TCP Congestion Avoidance Algorithm. *Technical note*. 1990.

[7] Dahlman, E. et al. The 3G Long-Term Evolution – Radio Interface Concepts and Performance Evaluation. *In Proceedings of Vehicular Technology Conference,* IEEE, 2006.

[8] Braden, B. et al, Recommendation on Queue Management and Congestion Avoidance in the Internet*, IETF RFC 2309*, April 1998.

[9] Allman, M. Paxson, V. Stevens, W. TCP Congestion Control. *IETF RFC2581*. April 1999.

[10] Stevens, W. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1994.

[11] 3GPP TS 36.302. Evolved Universal Terrestrial Radio Access (E-UTRA); Services provided by the physical layer, version 8.0.0. September 2007.

[12] 3GPP TS 36.401. Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Architecture description, version 8.2.0. June 2008.

[13] Mathis, M. Floyd, S. Romanow, A. TCP Selective Acknowledgment Options, *IETF RFC2018*. October 1996.

[14] Athuraliya, S. Low, S. Li, V. and Yin, Q. REM Active Queue Management. *IEEE Network Magazine*, May 2001.

[15] Yavuz, M. Khafizov, F. TCP over wireless links with variable bandwidth. *In Proceedings of Vehicular Technology Conference*, 2002.

[16] Xylomenos, G. Polyzos, G. C. TCP and UDP performance over a wireless LAN. *In Proceedings of InfoCom'99*, 1999.

[17] Park, K. Park, S. Park, D. Enhancing TCP performance over wireless network with variable segment size. *In Proceedings of PDPTA'01,* 2001.

[18] Ardelean, D. Blanton, E. Martynov, M. Remote Active Queue Management. *In Proceedings of NOSSDAV' 08.* Germany, 2008.

[19] Fall, K. Floyd, S. Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. *ACM SIGCOMM Computer Communication Review Volume 26, Issue 3.* ACM press, 1996.

[20] Sågfors, M. Ludwig, R. Meyer, M. Peisa, J. Queue Management for TCP Traffic over 3G Links. *Wireless Communications and Networking*, 2003.

[21] Sågfors, M. Ludwig, R. Meyer, M. Peisa, J. Buffer Management for Rate-Varying 3G Wireless Links Supporting TCP Traffic. *In Proceedings of Vehicular Technology Conference*, 2003.

[22] Sun, J. Ko, K.-T. Chen, G. Chan, S. Zukerman, M. PD-RED: To Improve the Performance of RED. *Communications Letters*. IEEE, 2003.

[23] 3GPP TS 36.321. Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) protocol specification, version 8.2.0. May 2008.

[24] 3GPP TS 36.322. Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Link Control (RLC) protocol specification, version 8.2.0. May 2008.

[25] 3GPP TS 36.323. Evolved Universal Terrestrial Radio Access (E-UTRA); Packet Data Convergence Protocol (PDCP) specification, version 8.2.1. May 2008.

[26] 3GPP TS 25.913. Requirements for Evolved UTRA (E-UTRA) and Evolved UTRAN (E-UTRAN), version 8.0.0. September 2008.

[27] Myung, H. G. Lim, J. Goodman, D. J. Peak-to-Average Power Ratio of Single Carrier FDMA Signals with Pulse Shaping. *In Proceedings of the 17th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*. Helsinki, Finland, September 2006.

[28] Lescuyer, P. Lucidarme, T. *Evolved Packet System: the LTE and SAE Evolution of 3G UMTS.* Wiley, 2007cification, version 8.2.0. May 2008.