Priority related QoS in TCP/IPv6 networks

This item was submitted to Loughborough University's Institutional Repository by the author.

Additional Information:

- A Doctoral Thesis. Submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of Loughborough University.

Metadata Record: https://dspace.lboro.ac.uk/2134/17329

Publisher: © Yunqiu Li

Rights: This work is made available according to the conditions of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0) licence. Full details of this licence are available at: https://creativecommons.org/licenses/by-nc-nd/4.0/

Please cite the published version.
<table>
<thead>
<tr>
<th>Author/Filing Title</th>
<th>LIY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class Mark</td>
<td>T</td>
</tr>
</tbody>
</table>

Please note that fines are charged on ALL overdue items.
Priority Related QoS in TCP/IPv6 Networks

Student: Yunqiu LI
ID Number: A479389

Supervisor: Prof. Shuang-Hua YANG

Loughborough University
Computer Science Department
November 2008
Submitted for the Degree of Doctor of Philosophy
Abstract

Active Queue Management (AQM) algorithms have been introduced for routers supporting TCP dataflows to assist the TCP congestion control mechanism to perform satisfactorily in all circumstances. However, today the Internet is expected to provide a distinctly different service to different users based on the value and content accessed, etc. The most popular AQM algorithm, Random Early Detection (RED), is famous for its fairness-addressed characteristic, which makes Quality of Service (QoS) differentiation impossible for it, and many other RED-based algorithms, while working on their own. The introduction of the IPv6 header flow labelling ability will allow new AQM algorithms which can provide differentiated levels of QoS based on user demands to be developed.

This thesis aims to develop a methodology for the AQM algorithms to work in conjunction with the IPv6 flow labelling ability, so as to provide acceptable Quality of Service (QoS) according to the end users' dynamically varying QoS requirements. We aim to offer a simple and generic process, which is adequately adoptable in Differentiated Services (DiffServ) TCP/IPv6 environments. The contribution of this thesis is four-fold. Firstly it develops a novel approach to allow AQM routers to track the QoS status of the dataflow and provide a differentiated QoS according to user requirement. Secondly, an AQM algorithm is developed from the classic Proportional Integral controller with the ability to support different QoS requirements with a faster response time. Thirdly, we have developed a mechanism to aid the AQM algorithm in predicting the queue length and improving system performance. Finally, we have proposed schemes to provide service differentiation for flows with different priorities in DiffServ Networks.
We demonstrate the proposed algorithms through simulations in Network Simulator 2 (NS2). The proposed methodologies are shown to exhibit many desirable properties, such as providing differentiated throughputs and fast system response. Compared with traditional AQM techniques, they offer significant improvements in guaranteeing different levels of QoS in TCP/IPv6 networks according to user requirements and the priority of the dataflows.
# Table of Contents

Abstract ................................................................................................................................. i
List of Publications .................................................................................................................. v
Acknowledgement .................................................................................................................... vi
List of Abbreviations .............................................................................................................. vii
List of Common Symbols ..................................................................................................... xii
List of Tables ......................................................................................................................... xvi
List of Figures ....................................................................................................................... xvi

Chapter 1 Introduction .......................................................................................................... I
  1.1 The Internet .................................................................................................................. 1
     1.1.1 IP Address Exhaustion and IPv6 ........................................................................ 2
     1.1.2 Internet Congestion ........................................................................................ 3
     1.1.3 The Increasing Demand of Quality of Service (QoS) ....................................... 5
  1.2 Research Motivation .................................................................................................... 8
  1.3 Objectives and Thesis Structure ............................................................................... 9

Chapter 2 DiffServ .................................................................................................................. 13
  2.1 DiffServ Architecture ................................................................................................. 13
  2.2 Functional Elements ................................................................................................. 13
     2.2.1 Classifiers ......................................................................................................... 13
     2.2.2 Profile ............................................................................................................... 14
     2.2.3 PHB .................................................................................................................. 15
  2.3 Classes ......................................................................................................................... 18
  2.4 Core Congestion Control .......................................................................................... 19

Chapter 3 TCP/IP and AQM .................................................................................................. 21
  3.1 TCP Congestion Control ............................................................................................ 21
     3.1.1 TCP Algorithms ............................................................................................... 21
     3.1.2 TCP-Based Protocols ...................................................................................... 26
  3.2 AQM and PQM .......................................................................................................... 28
  3.3 RED and ARED ......................................................................................................... 31
  3.4 Weighted Fair Queuing (WFQ) and Class-Based Weighted Fair Queuing (CBWFQ) .................................................................................................................. 38
  3.5 WRED and RIO ........................................................................................................ 41
  3.6 PI ............................................................................................................................... 44
  3.7 Conclusion .................................................................................................................. 46

Chapter 4 AQM Algorithms with Priority Checking Mechanism for TCP/IPv6 ............ 48
  4.1 RED with QoS Requirement Checking (QoSR-RED) .............................................. 49
     4.1.1 QoSR-RED algorithms .................................................................................... 49
     4.1.2 QoSR-RED NS Simulation ............................................................................ 51
  4.2 RED with QoS Alert Checking (QoSA-RED) ......................................................... 55
     4.2.1 Tagging algorithms at packet arrival ............................................................. 56
     4.2.2 Buffer management and packet dropping algorithms in QoSA-RED ............ 56
  4.3 QoSA-RED Modeling and Parameter Setting ......................................................... 58
     4.3.1 QoSA-RED Modeling .................................................................................... 58
     4.3.2 QoSA-RED PARAMETER SETTING ......................................................... 62
  4.4 QoSA-RED NS Simulation ....................................................................................... 73
4.5 Conclusion ........................................................................................................ 81

Chapter 5 Adaptive Priority Checking RED and Performance Analysis ...................... 82
5.1 Adapt QoSA-RED Parameters to network load level .................................................. 82
5.1.1 Adaptive QoSA-RED (AQA-RED) Algorithms .................................................... 83
5.1.2 AQA-RED NS Simulation .................................................................................. 84
5.2 RED with Gentle QoS Alert Checking (GQA-RED) ................................................ 89
5.2.1 Tagging algorithm at packet arrival .................................................................... 89
5.2.2 Buffer management and packet dropping algorithm in GQA-RED ................. 90
5.2.3 GQA NS Simulation ....................................................................................... 91
5.3 Conclusion ........................................................................................................... 95

Chapter 6 Proportional Integral Controller with Priority Checking Function ............... 97
6.1 Proportional Integral Controller for Queue Management ........................................ 97
6.2 Priority Checking PI Controller (PC-PI) ................................................................ 100
6.3 PC-PI NS Simulation ............................................................................................ 101
6.4 DSPCPI Controller .............................................................................................. 109
6.5 DSPCPI NS Simulation ....................................................................................... 111
6.6 Conclusion ........................................................................................................... 115

Chapter 7 Active Queue Management Algorithms with Prediction Mechanism ....... 117
7.1 Kalman Predictor .................................................................................................. 117
7.2 Kalman Equations for TCP/AQM ......................................................................... 120
7.3 Kalman PI Controller Algorithms .......................................................................... 122
7.4 Kalman PI Controller Simulation ........................................................................ 125
7.5 KPI Conclusion ..................................................................................................... 127

Chapter 8 Recommendations and Conclusion ............................................................. 129
8.1 Proposed Priority Checking AQM Algorithms ...................................................... 129
8.2 Network Architecture to enable PC-AQM Algorithms .......................................... 130
8.2.1 Overall Structure ............................................................................................ 130
8.2.2 Router Functionalities ..................................................................................... 132
8.3 Conclusion ........................................................................................................... 133

References ....................................................................................................................... 137
List of Publications


Acknowledgement

I would like to express my gratitude to all those who enabled me to complete this thesis. I want to thank the Department of Computer Science of Loughborough University for sponsoring my research. I am deeply indebted to my supervisor Prof. Shuang-Hua YANG, Dr. Roger Knott and my Director of Research Dr. Chris Hinde whose help, suggestions and encouragement throughout helped me the research for, and writing of, this thesis.

Furthermore I have to thank my colleagues, Fang YAO, Wu CHEN, Lei LIU, Khusvinder GILL, Yanning YANG and Dr. Jinghua ZHANG who inspired my research and encouraged me to go ahead with my thesis. I also want to thank Thomas G. Welch for proof reading my thesis more than once, correcting and offering suggestions for improving my English style and grammar.

Most of all, I would like to give my special thanks to my parents whose unconditional love and support enabled me to complete this work.
### List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK</td>
<td>Acknowledgement</td>
</tr>
<tr>
<td>ACL</td>
<td>Access Control List</td>
</tr>
<tr>
<td>ACP</td>
<td>Adaptive Control Protocol</td>
</tr>
<tr>
<td>AF</td>
<td>Assured Forwarding</td>
</tr>
<tr>
<td>AIMD</td>
<td>Additive Increase and Multiplicative Decrease</td>
</tr>
<tr>
<td>AQA-RED</td>
<td>Adaptive QoSA-RED</td>
</tr>
<tr>
<td>AQM</td>
<td>Active Queue Management</td>
</tr>
<tr>
<td>ARED</td>
<td>Adaptive Random Early Detection</td>
</tr>
<tr>
<td>AVQ</td>
<td>Adaptive Virtual Queue</td>
</tr>
<tr>
<td>BA</td>
<td>Behaviour Aggregate</td>
</tr>
<tr>
<td>BIC-TCP</td>
<td>Binary Increase Congestion Transmission Control Protocol</td>
</tr>
<tr>
<td>CBR</td>
<td>Constant Bit Rate</td>
</tr>
<tr>
<td>CBWFQ</td>
<td>Class-Based Weighted Fair Queuing</td>
</tr>
<tr>
<td>CIDR</td>
<td>Classless Inter-Domain Routing</td>
</tr>
<tr>
<td>CWND</td>
<td>Congestion Window</td>
</tr>
<tr>
<td>CWR</td>
<td>Congestion Window Reduced</td>
</tr>
<tr>
<td>DiffServ</td>
<td>Differentiated Services</td>
</tr>
<tr>
<td>DLCI</td>
<td>Data-Link Connection Identifier</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>DRED</td>
<td>Dynamic Random Early Detection</td>
</tr>
<tr>
<td>DS field</td>
<td>Differentiated Services Field</td>
</tr>
<tr>
<td>DSCP</td>
<td>DiffServ Code Point</td>
</tr>
<tr>
<td>DSPCPI</td>
<td>Double Status PCPI</td>
</tr>
<tr>
<td>DSRED</td>
<td>Double Slope Random Early Detection</td>
</tr>
<tr>
<td>DT</td>
<td>Drop Tail</td>
</tr>
<tr>
<td>ECN</td>
<td>Explicit Congestion Notification</td>
</tr>
<tr>
<td>EF</td>
<td>Expedited Forwarding</td>
</tr>
<tr>
<td>EWMA</td>
<td>Exponentially Weighted Moving Average</td>
</tr>
<tr>
<td>FEC</td>
<td>Forwarding Equivalence Class</td>
</tr>
<tr>
<td>FIFO</td>
<td>First In First Out</td>
</tr>
<tr>
<td>FRED</td>
<td>Flow Random Early Detection</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>GQA-RED</td>
<td>Gentle QoSA-RED</td>
</tr>
<tr>
<td>GRED</td>
<td>Gentle Random Early Detection</td>
</tr>
<tr>
<td>HSTCP</td>
<td>High-Speed Transmission Control Protocol</td>
</tr>
<tr>
<td>H-TCP</td>
<td>High delay-bandwidth Transmission Control Protocol</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>IntServ</td>
<td>Integrated Services</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Full Form</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IPv6</td>
<td>Internet Protocol Version 6</td>
</tr>
<tr>
<td>ISP</td>
<td>Internet Service Provider</td>
</tr>
<tr>
<td>KPI</td>
<td>PI controller with Kalman Predictor</td>
</tr>
<tr>
<td>LER</td>
<td>Label Edge Router</td>
</tr>
<tr>
<td>LSP</td>
<td>Label Switch Path</td>
</tr>
<tr>
<td>LSR</td>
<td>Label Switch Router</td>
</tr>
<tr>
<td>MF</td>
<td>Multi-Field</td>
</tr>
<tr>
<td>MPLS</td>
<td>Multi-Protocol Label Switching</td>
</tr>
<tr>
<td>NS</td>
<td>Network Simulator</td>
</tr>
<tr>
<td>PC</td>
<td>Priority Checking</td>
</tr>
<tr>
<td>PCPI</td>
<td>Priority Checking Proportional Integral controller</td>
</tr>
<tr>
<td>PD</td>
<td>Proportional Differential</td>
</tr>
<tr>
<td>PDA</td>
<td>Personal Digital Assistant</td>
</tr>
<tr>
<td>PHB</td>
<td>Per-Hop Behaviour</td>
</tr>
<tr>
<td>PI</td>
<td>Proportional Integral</td>
</tr>
<tr>
<td>PID</td>
<td>Proportional Integral Differential</td>
</tr>
<tr>
<td>PQM</td>
<td>Passive Queue Management</td>
</tr>
<tr>
<td>PRC</td>
<td>Proportional Rate-Based Control</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>QoSA-RED</td>
<td>QoS Alert aware RED</td>
</tr>
<tr>
<td>QoSRED</td>
<td>QoS Requirement aware RED</td>
</tr>
<tr>
<td>RCP</td>
<td>Rate Control Protocol</td>
</tr>
<tr>
<td>RED</td>
<td>Random Early Detection</td>
</tr>
<tr>
<td>REM</td>
<td>Random Exponential Marking</td>
</tr>
<tr>
<td>RFC</td>
<td>Request for Comments by IETF</td>
</tr>
<tr>
<td>RIO</td>
<td>Random Early Detection with In / Out bit</td>
</tr>
<tr>
<td>RTT</td>
<td>Round Trip Time</td>
</tr>
<tr>
<td>RWND</td>
<td>Receiver's advertised Window</td>
</tr>
<tr>
<td>SACK</td>
<td>TCP Selective Acknowledgement</td>
</tr>
<tr>
<td>RED</td>
<td>Stabilized Random Early Detection</td>
</tr>
<tr>
<td>SSTRESH</td>
<td>Slow Start Threshold</td>
</tr>
<tr>
<td>STCP</td>
<td>Scalable Transmission Control Protocol</td>
</tr>
<tr>
<td>TCA</td>
<td>Traffic Conditioning Agreement</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TOS</td>
<td>Type of Service</td>
</tr>
<tr>
<td>TSW</td>
<td>Time Sliding Window</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>VoIP</td>
<td>Voice over IP</td>
</tr>
<tr>
<td>VRC</td>
<td>Virtual Rate Control</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>WFQ</td>
<td>Weighted Fair Queuing</td>
</tr>
<tr>
<td>WRED</td>
<td>Weighted Random Early Detection</td>
</tr>
<tr>
<td>XCP</td>
<td>eXplicit Control Protocol</td>
</tr>
</tbody>
</table>
List of Common Symbols

\( a_{pl} \): PI-AQM controller parameter

\( a \): \( p_{\text{max}} \) updating parameter in AQA-RED

\( a_{j} \): Average dropping rate

\( b_{pl} \): PI controller parameter

\( b \): \( p_{\text{max}} \) updating parameter in AQA-RED

\( C \): Link capacity

\( c \): \( p_{\text{max}} \) updating parameter in AQA-RED

\( c(n) \): Value of CWND at time \( n \)

\( \text{count} \): The number of incoming packets from the last dropping till the current incoming packet.

\( C(s) \): Transfer function of controller

\( C_{\text{red}}(s) \): RED transfer function

\( C_{p}(s) \): Transfer function of High-Priority dataflow controller

\( C_{n}(s) \): Transfer function of Non-Priority dataflow controller

\( f_{\text{id}} \): Flow ID

\( k \): Sample instant

\( K_{p} \): Proportional factor
$K_i$: Integral factor

$K_d$: Derivative factor

$K_{pi}$: PI controller parameter

$L$: Average dropping rate limit

$\text{min}_{sh}$: Minimum threshold of the queue size

$\text{max}_{sh}$: Maximum threshold of the queue size

$N$: Network work load

$N^-$: Lower bound of network work load

$N_p$: Number of active High-Priority dataflows

$N_n$: Number of active Non-Priority dataflows

$n_a$: Number of arrival packets

$n_d$: Number of dropped packets

$p$: Dropping probability

$p_{\text{max}}$: Maximum dropping probability

$p_n$: Dropping probability of Non-Priority dataflow

$p_p$: Dropping probability of High-Priority dataflow

$P_{\text{red}}$: Dropping probability generated by RED

$p_{pi}$: Dropping probability generated by PI-AQM
\( p_{pc-pl} \): Dropping probability generated by PC-PI

\( p(n) \): The number of packets sent at time \( n \)

\( P(s) \): Transfer function of plant

\( P_{tcp}(s) \): Transfer function of TCP plant

\( P_{queue}(s) \): Transfer function of Queue plant

\( q \): Queue size.

\( q_0 \): Desired queue size

\( q_a \): Current QoS status

\( q_{ave} \): Average queue size

\( q_{cur} \): Current queue size

\( q_{ref} \): Target queue size

\( q_{ref2} \): Target queue size under Critical control status in DSPCPI

\( q_e \): QoS Requirement

\( R \): Round trip time

\( R_0 \): Time delays at steady state

\( R^* \): Upper bound of time delays

\( r_p \): Ratio of High-Priority flows to the overall work load

\( r_{p_{max}} \): Maximum \( r_p \)
$S_{up}$ : Update status

$T$ : Sampling time of PI controller

$T_p$ : Propagation delay in seconds

$w$ : Weight factor for EWMA

$W$ : Expected TCP window size in packets

$X$ : The amount of dropped package

$\alpha$ : Queue weight, usually set as 0.002 / increment factor used in ARED

$\beta$ : Decrease factor used in ARED

$\delta p$ : Fluctuation of dropping probability

$\delta q$ : Fluctuation of queue size

$\omega_g$ : Low frequency threshold
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Assured Forwarding (AF) Behaviour Group</td>
<td>17</td>
</tr>
<tr>
<td>Table 2</td>
<td>QoSR-RED Priority List Entry Structure</td>
<td>50</td>
</tr>
<tr>
<td>Table 3</td>
<td>QoSA-RED Priority List Entry Structure</td>
<td>55</td>
</tr>
</tbody>
</table>

List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>IPv4 Packet Header Structure</td>
<td>2</td>
</tr>
<tr>
<td>Figure 2</td>
<td>IPv6 Packet Header Structure</td>
<td>2</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Logical view of DiffServ Functional Elements</td>
<td>13</td>
</tr>
<tr>
<td>Figure 4</td>
<td>TCP CWND Updating Rules in Congestion Avoidance Phase</td>
<td>23</td>
</tr>
<tr>
<td>Figure 5</td>
<td>TCP CWND Updating Rules in Slow Start Phase</td>
<td>24</td>
</tr>
<tr>
<td>Figure 6</td>
<td>TCP Congestion Window Updating Rules in general</td>
<td>24</td>
</tr>
<tr>
<td>Figure 7</td>
<td>TCP Congestion Window increases in different periods</td>
<td>24</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Duplicate ACKs</td>
<td>25</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Fast Retransmit and Fast Recovery Algorithms</td>
<td>27</td>
</tr>
<tr>
<td>Figure 10</td>
<td>RED Dropping Probability to The Average Queue Size</td>
<td>33</td>
</tr>
<tr>
<td>Figure 11</td>
<td>RED Algorithm</td>
<td>34</td>
</tr>
<tr>
<td>Figure 12</td>
<td>ARED Algorithm</td>
<td>37</td>
</tr>
<tr>
<td>Figure 13</td>
<td>New ARED Algorithm</td>
<td>38</td>
</tr>
<tr>
<td>Figure 14</td>
<td>WFQ Structure</td>
<td>39</td>
</tr>
<tr>
<td>Figure 15</td>
<td>WRED Structure</td>
<td>42</td>
</tr>
<tr>
<td>Figure 16</td>
<td>RIO TSW Algorithm</td>
<td>43</td>
</tr>
<tr>
<td>Figure 17</td>
<td>RIO Marking Algorithm</td>
<td>43</td>
</tr>
<tr>
<td>Figure 18</td>
<td>RIO Structure</td>
<td>44</td>
</tr>
<tr>
<td>Figure 19</td>
<td>PI AQM Block Diagram</td>
<td>45</td>
</tr>
<tr>
<td>Figure 20</td>
<td>PI AQM Algorithms</td>
<td>46</td>
</tr>
<tr>
<td>Figure 21</td>
<td>PC-AQM Generic Structure</td>
<td>49</td>
</tr>
</tbody>
</table>
Figure 22 QoSR-RED Tagging Algorithm ........................................................... 50
Figure 23 QoSR-RED Buffer Management and Packet Dropping Algorithm...... 51
Figure 24 Single Bottle-Link Topology ................................................................. 51
Figure 25 Queue Length of RED and QoSR-RED in Simulation 1 ...................... 52
Figure 26 Throughput of FTP Flows under RED during Simulation 1 .............. 54
Figure 27 Throughput of FTP Flows under QoSR-RED during Simulation 1 ..... 54
Figure 28 QoSA-RED Tagging Algorithm ........................................................... 57
Figure 29 QoSA-RED Queue Management and Dropping Algorithm .......... 57
Figure 30 Simplified TCP-queue Dynamics ......................................................... 59
Figure 31 TCP/AQM Block Diagram ................................................................... 60
Figure 32 TCP/PC-RED Block Diagram .............................................................. 60
Figure 33 TCP/PC-RED Block Diagram with P’n(s) as Control Plan .............. 61
Figure 34 Simplified Block Diagram of PC-RED with TCP Sources .............. 61
Figure 35 Region of N to use Nyquist Criterion in C-N Plane .......................... 64
Figure 36 Region of N to use Nyquist Criterion in C-N Plane when N < 300 .... 64
Figure 37 Nyquist Plot of G(s) with various r_p .................................................. 66
Figure 38 Bode Diagram of G(s) with 0.005 as r_p .......................................... 66
Figure 39 Bode Diagram of G(s) with 0.995 as r_p .......................................... 67
Figure 40 a_d as the function of the weight factor w and the packet dropping counter n .......................................................... 73
Figure 41 Queue Length of RED, QoSR-RED and QoSA-RED in Simulation 2 75
Figure 42 Throughput of High-Priority flow and Non-Priority flow in Simulation 2 ................................................................................ 76
Figure 43 Throughputs and Average Dropping Rates under Different w ......... 77
Figure 44 Average Dropping Rates and Throughputs of High-Priority and Non-Priority Flows when r_p = 10% .................................................. 79
Figure 45 Average Dropping Rates and Throughputs of High-Priority and Non-Priority Flows when r_p = 30% .................................................. 80
Figure 46 AQA-RED Maximum Dropping Probability Updating Algorithm ..... 83
Figure 47 Queue size under QoSA-RED ............................................................. 85
Figure 48 Average Dropping Rate of Flow i and Flow j ................................... 86
Figure 78  Queue size controlled by PI controller with time delays ...................... 122
Figure 79  Queue size controlled by PI controller without time delays ............... 123
Figure 80  Generic Time Delay System Block Diagram ..................................... 123
Figure 81  Kalman PI Controller Structure .......................................................... 124
Figure 82  TCP/AQM with PI and KPI Controller ............................................... 125
Figure 83  TCP/AQM with PI and KPI Controller ............................................... 126
Figure 84  TCP/AQM with PI and KPI Controller ............................................... 127
Figure 85  PC-AQM Components and Functions ............................................... 129
Figure 86  Network Components ........................................................................ 131
Figure 87  Router to Router ................................................................................ 131
Figure 88  Network to Network ........................................................................... 132
Figure 89  Router Functions with Fair Queuing-Based Queuing Techniques .... 133
Chapter 1 Introduction

1.1 The Internet

From the 1950s, the Internet has grown out of a group of communication networks limited by their nature, only allowing them to communicate between stations on the network, into a worldwide series of interconnected networks which has become an essential part of the global communication infrastructure, transmitting data by packet switching using Internet Protocol (IP) [1]. It now consists of millions of smaller academic, business, domestic and government networks. These networks together carry various information and services, such as e-mail, online video and audio conference, file transfer, stock trading, industrial real-time control and the World Wide Web. The number of the Internet users rose to one billion in the first 36 years of its development. Since then, the Internet use has grown at 18% per year. NUA Internet Surveys, a world-renowned online resource providing quantitative and qualitative information on global Internet, has announced in 2002 that there were more than 605 million internet users worldwide [2]. Based on 2002 Nielsen NetRatings, which is a service defining the global standard in online audience measurement, there were over 34 million active users in the UK. According to a People's Daily report, the global average Internet penetration rate is 17.6% of 20th July 2007. In some countries with more developed Internet construction such as the United States, Japan and South Korea, the penetration rate even exceeds 65%. And in some developing countries, such as China, although the penetration rate is only 12.3%, the annual growth rate of Internet users reaches 31.7%. It is believed that the second billion Internet users will probably be added by 2015.

At this new stage of rapid growth in both breadth and depth, the Internet is facing a number of challenges. These challenges have resulted in increased research being undertaken in a number of relevant fields listed as follow.
1.1.1 IP Address Exhaustion and IPv6

IP is a data-oriented protocol on the network layer and is encapsulated in a data link layer protocol (e.g., Ethernet). The IP most widely implemented today is IPv4 [3][4]. As a network layer protocol, it provides communicable unique global addressing service amongst computers. However, as the demand of information transmission is growing tremendously, it is anticipated that in the near future, its usage is expected to extend to not only computers, but PDAs, phones, and even home appliances, such as televisions, refrigerators, washing machines, and perhaps even coffee makers. This certainly would lead to a decreasing availability of public IP addresses. This concern has spanned decades as the first concerns were raised in the 1980s. Research on several feasible solutions has taken place, such as Classful Networks [5], Classless Inter-Domain Routing (CIDR) addressing [6], and IPv6 [7], which is produced by the Internet Engineering Task Force (IETF).

![IPv4 Packet Header Structure](image1.png)

**Figure 1** IPv4 Packet Header Structure

![IPv6 Packet Header Structure](image2.png)

**Figure 2** IPv6 Packet Header Structure
Since 2007, IPv6 has been widely considered as the only long-term solution for IP address exhaustion. But even as the deadline for IPv4 address exhaustion approaches, most Internet Service Providers and Equipment Vendors are only just starting to consider the widespread deployment of IPv6. Therefore, it is being adopted very slowly. IPv6 is a major revamp to the IPv4 architecture, such as addressing performance, scalability, security. Figure 1 and Figure 2 show the differences between IPv4 and IPv6 header structures. The main improvement is a much larger address space, which jumps from 32-bit format to 128-bit so as to allow greater flexibility in assigning addresses. By implementing IPv6, the Internet is able to support $2^{128}$ addresses, which is approximately $5.24 \times 10^{28}$ for each of the roughly 6.5 billion people alive today. The major intention of IPv6 is to give permanent unique addresses to every individual device that can be connected to the Internet. It also eliminates the need to use network address translation to avoid address exhaustion, and additionally simplifies aspects of address assignment and re-numbering when changing providers.

1.1.2 Internet Congestion

In May 1974, Vinton Cerf and Robert Kahn proposed the first draft of TCP/IP [8]. In December, Vinton Cerf and other researchers published the first technical specification of TCP/IP as an Internet Experiment Note [9]. Since then the network has been continuously developed and deployed. Although Internet Congestion Collapse was identified as a potential problem as far back as 1984 [10], the Internet experienced its first collapse in October 1986. The link from LBL to UC Berkeley is a 400-yard-long National Science Foundation Network (NSFnet) phase-I backbone consisting of 3 hops. During that period, the data throughput dropped from 32 Kbps to 40bps. Researchers were fascinated by the sudden three orders of magnitude drop in bandwidth and embarked on an investigation, searching for causes and solutions. Two years later, the first Congestion Control Algorithm, TCP Flow Control [11], was proposed by Van Jacobson. Since then a great deal of research has been performed on that subject.
Chapter 1 Introduction

At present, Congestion Control comprises two main parts, the Source Algorithms, such as Transmission Control Protocol (TCP) [12] and Explicit Control Protocol (XCP) [13], and the Link Algorithms, such as Active Queue Management (AQM) [14].

**Congestion Control Source Algorithms**

The TCP congestion control mechanism is an Additive Increase and Multiplicative Decrease (AIMD) mechanism, consisting of four basic functions: Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery. It is considered to be the primary basis for much of the research on Source Algorithms. HighSpeed TCP (HSTCP) [15], which is defined in RFC3649 in December 2003, is a modification of TCP’s congestion control mechanism for use with TCP connections with large congestion windows. BIC TCP (Binary Increase Congestion control) [16] is an implementation of TCP with an optimized congestion control algorithm for high speed networks with high latency, which is implemented by default in Linux kernels. XCP controls the dynamics of the aggregate traffic independently from the relative throughput of the individual flows in the aggregate so as to stabilise the source node performance under high-bandwidth and large-delay links. Rate Control Protocol (RCP) [17] is designed to be a practical way to emulate processor sharing, and allows flows to complete much faster than with TCP or XCP under a broad range of conditions. ACP [18] is an Adaptive Control Protocol with a learning capability. This capability enables the protocol to adapt to dynamically changing network conditions and maintain stability. Like TCP, ACP is a window based protocol. But it does not require the maintenance of per flow states within the network.

**Congestion Control Link Algorithms**

The Link Algorithms are mainly focused on Queue Management. As one of the Passive Queue Management (PQM) methods, Drop-Tail [19] was widely used due to its simplicity and fairness. However, it can cause TCP global synchronization during periods of congestion because each sender reduces their transmission rate at the same time when packet loss occurs. This problem has been the subject of much research. Several AQM algorithms have been proposed for the link to give notice to the source informing it of the
congestion beforehand so as to reduce the likelihood of global synchronisation, as well as to keep the queue size small in the face of heavy loads and burst traffics.

The most widely deployed AQM algorithm is Random Early Detection (RED) [20]. It is believed to have much better performance than the Drop-Tail. However, due to its complex parameterization and algorithmic limitation, RED could lead to instability and low frequency oscillations in the regulated output. RED has been investigated by many researchers [21] [22] [23]. Several tuning methods have been proposed [24] [25] [26]. There have also been some new AQM algorithms developed based on RED, such as Adaptive RED (ARED) [27], Weighted RED (WRED) [28], RED with In/Out bit (RIO) [29], Double Slope RED (DSRED) [30], Stabilized RED (SRED) [31] and Dynamic RED (DRED) [32]. Other AQM algorithms that have been proposed in the recent years also have their advantages in certain respects, such as Random Exponential Marking (REM) [33], Adaptive Virtual Queue (AVQ) [34], Virtual Rate Control (VRC) [35] and Proportional Rate-Based Control (PRC) [36]. When analyzing in the System Control Theory perspective, the TCP/AQM infrastructure can be considered to be a feedback control system, and more algorithms have been determined, such as Proportional-Integral (PI) AQM [37] and Proportional-Differential (PD) AQM [38]. However, more work is needed to get them ready for practical deployment.

1.1.3 The Increasing Demand of Quality of Service (QoS)

Although the research on Source Algorithms and Link Algorithms are well-developed and are continuously under active development, and some have already been successfully put into practise, a study of Internet performance reveals that user experience is still highly variable and often poor. Therefore, new requirements for network infrastructure have been generated. People would like to see QoS approaches implemented by applications, servers, network routers or switches, to prioritize Internet traffic and transactions across the entire infrastructure. More specifically, the Internet is expected to provide a distinct service to different users based on value, and to deliver a distinct service based on content accessed, etc.
QoS could be considered to be a Resource Reservation Method which provides different priorities to different users, or guarantees a certain level of performance for a data flow according to requests from application or the ISP policy. However, it can be used as a quality measurement with many alternative definitions, rather than referring to the ability to reserve resources. QoS covers all the aspects of a connection, such as time to provide service, transmission quality, loss, reliability, etc. When the network capacity is limited, for example in cellular data communication, especially for real-time streaming multimedia applications, such as voice over IP (VoIP) and IP-TV, since these often require fixed bit rate and are delay sensitive, QoS guarantees would be very important.

- **Integrated Services and Differentiated Services**

  The Best-effort Protocol, which is to some extent deserted by the Internet, does not support QoS. Presently, IP routers are able to provide mechanisms for differentiated or guaranteed QoS to certain data flows, based on Integrated Services (IntServ) or Differentiated Services (DiffServ) Protocols. IntServ specifies the fine-grained QoS system, which is distinguished from a coarse-grained control system specified in DiffServ. Assuming that every router in the network works with IntServ, by making individual reservation for every application that requires some kind of guarantee, the IntServ can provide QoS guarantees to those applications with QoS needs. However, since the reservation making needs "Flow Specs", which describes what the reservation is for, as guidance, a very large number of state information must be stored in each router. As a result, IntServ can only work on small-scale Internet. Otherwise, it is difficult to keep track of all of the reservations. Unlike IntServ, a flow-based mechanism, DiffServ is a class-based mechanism, which operates under the principle of traffic classification, where every data packet is assigned a limited number of traffic classes, rather than every individual flow. Every router in the network is configured to differentiate traffic based on its class. Every traffic class can be managed differently, so as to ensure preferential treatment for high-priority traffic in the network. The DiffServ model does not specify how to assign the priorities. It simply recommends a standardized set of traffic classes and provides a framework to allow the classification and differentiated treatment of these
traffics with different priorities. DiffServ needs to rely on a separate mechanism to classify the packets. All the traffic being transferred through a router that belongs to the same class is referred to as a Behaviour Aggregate (BA). Then the DiffServ-aware routers working with DiffServ can implement Per-Hop Behaviour (PHB), which defines the packet forwarding method associated with a particular class of traffic. Different PHBs may be defined to offer, for example, low-loss and low-latency forwarding methods or best-effort forwarding methods.

Today, DiffServ has widely superseded other Network Layer QoS mechanisms, such as IntServ, as the primary protocol implemented in routers to provide different levels of service. The advantage of DiffServ is that it simplifies the service mechanisms inside the core of the Internet, as all the policing and classifying is done at the boundaries between DiffServ clouds. Based on the DiffServ operating approach, the research on DiffServ is mainly focused on Traffic Classification, the Header Marking and the Multi-Class Scheduling.

- Multi-Protocol Label Switching

Multi-Protocol Label Switching (MPLS) [44] is also a popular mechanism providing potential ability for the ISPs to provide QoS while using it in conjunction with other network mechanisms, such as blocking mechanisms, etc. The operating approach of MPLS is similar to DiffServ in some scales. It has a Label Edge Router (LER) to analyse the packets and to determine which Forwarding Equivalence Class (FEC) it belongs to and the Label Switch Path (LSP) of the packets, just like the Classification and BA or PHB in DiffServ. In DiffServ the DS field in the IP header would be marked with the class, in MPLS the packet would be labelled with a MPLS header. The difference between these two techniques is that MPLS focuses on routing while DiffServ is about queuing, scheduling and dropping. In a MPLS network the packet header is only examined once at the edge for the LER to generate the MPLS label, which is then checked at every Label Switch Router (LSR) in the network, while in DiffServ network the packet header is analysed every hop. Attempting to synergise MPLS and DiffServ has been popular research topic as MPLS and DiffServ appear to be independent of each other, and they can both provide higher QoS in different forms.
Chapter 1 Introduction

QoS-Routing

QoS-Routing [45] is an important component in the whole QoS framework for the Internet. Unlike the Best-Effort routing protocols, which use single objective optimization algorithms considering only one metric such as the bandwidth, hop count or the cost of the network, QoS-Routing is a mechanism “under which paths for flows are determined based on some knowledge of resource availability in the network as well as the QoS requirement of the flows.” [46] It is a part of traffic engineering which makes the traffic engineering process automatic. Implementing QoS-Routing together with DiffServ or MPLS can help to solve some potential congestion problems in the latter two. Since using these latter approach the packets are marked only at the edge of the network, and the routers in the core network only provide corresponding services depending on the marks, there is a possibility that a core link would be congested with high numbers of packets with the same marks. QoS-Routing can help to check the link status beforehand so as to avoid potential congestion.

1.2 Research Motivation

From what has been mentioned above we can reach the conclusion that most of the QoS mechanisms can be thought of as the composition of analyzing the QoS requirements, which mainly are marked in the packet headers, and providing available network layer services, such as routing, queuing and scheduling. At the moment when IPv6 is replacing IPv4 to solve the IP address exhaustion problem, it also provides a new IP header which contains fewer components but give a greater capacity to the packet. This offers an opportunity to improve the QoS mechanism of the Internet by providing more specific services to packets whose IPv6 header contains special flow labels.

This thesis aims to develop a methodology for AQM algorithms to work with the IPv6 flow labelling ability to provide an acceptable QoS according to the end users' dynamically varying QoS requirements. We aim to offer a simple and generic process, which makes it adequately adoptable in DiffServ TCP/IPv6 environments.
1.3 Objectives and Thesis Structure

The objectives of my research include the following aspects.

1. **Easily adoptable mechanism to bring much more meticulous QoS consideration to existing AQM algorithms.** The flow labeling ability of IPv6 allows the end users QoS Requirement to be defined in much greater detail. Hence the Internet will face more challenges to support these meticulous requirements and helping the end users to achieve what they ask for. Since the Internet has already become an enormous global communication system consisting of numerous well-developed mechanisms achieved through decades of improvement, it is necessary for us to develop an easily adoptable mechanism to work with existing network components rather than something more complicated.

2. **Protection for Non-Priority flows under heavy congestion.** The main characteristic shared by most of the router algorithms with some QoS consideration, irrespective of queue management or scheduling, is undermining the QoS status of Non-Priority flows to guarantee higher QoS status for flows with QoS Requirements under congestion. When the congestion gets more severe, throughputs of Non-Priority flows will face a great probability of delay. It is needed to protect the throughputs of Non-Priority flows under heavy congestion without decreasing the QoS status of the High-Priority flows. In addition, sometimes under light congestion, Non-Priority flows can still suffer from low throughput due to priority related queue management algorithms. It is important to ensure that Non-Priority flows do not suffer from unnecessary and excessive discrimination.

3. **Reducing the response time of the Priority-aware TCP/AQM control system.** When analyzing system performance, response time and fluctuation are the main aspects. Inappropriate parameterization of
RED algorithms can make the system quite sensitive to the change of load level. Even if a traditional PI controller is used instead of RED, there is still a parameter tuning problem [37]. When introducing a Priority Checking mechanism to AQM algorithms, the algorithms become computationally expensive. It is important to reduce the system response time without adding too much extra computation.

4 Making up for the time delays caused by the Priority Checking and Packet Labeling actions. The priority checking mechanism brings extra computation into the system which can cause longer delays. The queuing delay may cause large oscillation of the queue size, which ultimately leads to unstable performance. For small-scale network with small portion of High-Priority flows, the amount of computation may not be observed. However, for a larger sized network or a network with large portion of High-Priority flows, it is necessary for us to develop an approach to make up for the extra time delays caused by the Priority Checking mechanism.

5 Recommendation of the network configuration to support such Priority Related AQM algorithms. The current Internet infrastructure is all pervasive and consequently any proposal for revampment would not be successful. As, the objective of using IPv6 flow labeling ability is to trigger different levels of services provided by the Internet, the problem is how can we minimize the adjustment and enable the Internet to guarantee the end users demands to the greatest extent.

In this thesis, we have presented seven AQM algorithms, which are developed to gradually achieve the research objectives listed above. The contributions and structure of the thesis is as follow.

Chapter 2 and Chapter 3 introduce the DiffServ environment and the related works on Congestion Control.
Chapter 4 presents our first two Priority Checking AQM (PC-AQM) algorithms, QoS Requirement aware RED (QoSR-RED) and QoS Alert aware RED (QoSA-RED). The contribution of QoSR-RED is two-fold. Firstly, it allows the existing RED algorithm to respond dissimilarly according to meticulous dataflow QoS Requirement information. Secondly, it is easy to adopt by other AQM algorithms. QoSA-RED is developed from QoSR-RED. It inherits the contributions of QoSR-RED, and has the ability to protect dataflows without QoS Requirement under heavy congestion.

Chapter 5 presents two adaptive versions of QoSA-RED, Adaptive QoSA-RED (AQA-RED) and Gentle QoSA-RED (GQA-RED). AQA-RED has further contribution on protecting Non-Priority dataflows while guaranteeing certain level of QoS status to dataflows with high QoS Requirement under heavy congestion. On the other hand, GQA-RED inherits the contributions of QoSA-RED, and has the ability of tracking the QoS status throughout transmission so as to protect certain level of QoS status to High-Priority dataflows in multi-congested-link networks.

In Chapter 6, we have combined PI-AQM controller and GQA-RED, and developed Priority Checking PI controller (PCPI) to achieve a smaller system response time. We have also developed Double Status PC-PI (DSPCPI) controller, which has the ability to remove the overshoot, so as to stabilize the QoS status of both High-Priority and Non-Priority dataflows when network load level varies.

To compensate for the time delay caused by the Priority Checking function, a PC-PI controller with the Kalman Predictor (KPI) is proposed in Chapter 7. KPI not only inherits the contribution from the previous PC-AQM algorithms, it can also make up for the time delays caused by extra computation required for the priority checking and packet labelling, etc.

These PC-AQM algorithms consist of traditional AQM algorithms and extra components developed for IPv6 networks: Priority Checker, Packet Labeller, Parameter Adapter and Queue Size Predictor. These components can work both individually and in conjunction with one or more than one other components. If the end users have difficulty labelling their QoS Requirements, the Internet Service Providers can always specify the
requirement for them and label the incoming or outgoing packets at routers configured with these components. In Chapter 8, we summarize the network architecture requirement to support our proposed algorithms.
Chapter 2  DiffServ

2.1 DiffServ Architecture

DiffServ is a service that defines some significant characteristics of packet transmission in one direction across a set of one or more paths within a network. It is based on a simple model where traffic is classified upon entering the network and subject to possible conditioning at the boundaries of the network before being assigned to different Behaviour Aggregates (BA).

![Logical view of DiffServ Functional Elements](image)

2.2 Functional Elements

The architecture of DiffServ, as shown in Figure 3, consists of several functional elements, including Per-Hop forwarding Behaviours (PHB), packet classification functions and traffic conditioning functions, such as metering, marking, shaping and policing. These complex functions elements, such as the classifiers and traffic conditioners, are implemented at the network edge nodes, so as to leave the transmission in the core network as simple as possible by applying corresponding PHB to aggregated traffics.

2.2.1 Classifiers

Network packets entering a DiffServ network are subject to classification and conditioning. Packets in these traffic streams are selected by the Packet Classifiers based on the content of some fields of its header. There are two types of classifier. Each of them classifies traffic by different parameters. The Behaviour Aggregate Classifier
Chapter 2 DiffServ
categorizes packets based on the DS \[47\] \[48\] code point, which is further explained in later sections, in the IP header only. The Multi-Field (MF) classifier selects packets based on the value of a combination of one or more header fields, such as protocol ID, DS field, source address, destination address, source port and destination port numbers, and other information such as incoming interface.

Traffic classifiers \[49\] may respect any DiffServ markings carried by the packets or may choose to ignore or override them. As network operators want tight control over the type and volumes of traffic in a given class, there is only a small probability that the network will respect those markings at the entrance to the DiffServ network. Traffic in each class may be further conditioned by subjecting the traffic to a rate limiter, traffic policer or shaper. Packets matching specific criteria are steered by the classifier to an element of a traffic conditioner for further processing. Some management procedures are needed to configure the classifiers in accordance with the appropriate Traffic Conditioning Agreement (TCA).

2.2.2 Profile

A traffic profile is an optional component of a TCA, the usage of which is dependent on the specifics of the offered service and the network service provisioning policy. A traffic profile specifies the temporal properties of a traffic stream selected by a classifier. It provides rules to decide whether a particular packet is in or out of a conformity profile. For example, a profile based on a token bucket may stipulate that all packets marked with DS code point D should be measured against a token bucket meter with rate $R$ and burst size $B$.

**Profile: codepoint = D, use token-bucket R, B**

In this case, out-of-profile packets are those in the traffic stream which arrive when insufficient tokens are available in the bucket. The concept of in-profile and out-of-profile can be extended to more than two levels, such as multiple levels of conformity with a profile may be defined and enforced.
Different conditioning actions may be applied to the in-profile packets and out-of-profile packets respectively, or different accounting actions may be triggered. Out-of-profile packets may be mapped to Behaviour Aggregates which are inferior in some aspects of forwarding performance to the BA into which in-profile packets are mapped. In-profile packets may enter the DiffServ network without further conditioning. Alternatively, the DS field in their headers may be changed. The latter happens when the DS code point is set to a non-Default value for the first time, or when the packets enter a DiffServ network that uses a different PHB group or code point-to-PHB mapping policy for this traffic stream. Out-of-profile packets may be queued until they are in-profile, discarded, marked with a new DS code point, or forwarded unchanged while triggering some accounting procedure.

2.2.3 PHB

A Per-Hop Behaviour (PHB) is the means by which a node allocates resources to traffic aggregates, and it is on top of the basic hop-by-hop resource allocation mechanism from which differentiated services may be constructed. PHBs are implemented in nodes by means of buffer management mechanisms on a network node’s output interface queue. PHB implementations are recommended to avoid any packet re-ordering within a flow.

The PHB applied to aggregated traffic is associated with a 6-bit Differentiated Services Code Point (DSCP) in the 8-bit DS field of the current IP packet header. The DS field is the same as the TOS field in IPv4 old version, and ECN occupies the upper 2 bits. Theoretically, a network could have up to 64 different traffic classes using different markings in the DSCP. The DiffServ RFCs recommend, but do not require, certain encodings so as to provide great flexibility in defining traffic classes to the network operator. However, in practice, most networks use commonly-defined Per-Hop Behaviours as follows:

Default PHB: for typically best-effort traffic

Expedited Forwarding (EF) PHB: for low-loss, low-latency traffic

Assured Forwarding (AF): behaviour group
Class Selector PHB: defined to maintain backward compatibility with the IP Precedence field.

**Default PHB**

This is the only required behaviour. Basically, any traffic that does not satisfy the requirements of any other defined class is mapped to the default PHB. Usually, the default PHB has best-effort forwarding characteristics.

**Expedited Forwarding (EF) PHB**

The EF PHB ensures that the traffic aggregates are delivered at certain configured rates. During the connection setup, a Service Level Agreement is used to ensure that the departure rate of the aggregate’s packets from any DiffServ node should equal or exceed a configurable rate. The EF traffic should receive this rate independent of the intensity of any other traffic attempting to transit the node. An overload of EF traffic will cause queuing delays and affect the jitter and delay tolerances within the class. Thus the EF PHB should be strictly controlled through admission control, policing and other mechanisms, so as to minimize the damage that EF traffic would cause to other traffic aggregates, if it is implemented by a mechanism that allows unlimited pre-emption of other traffic.

Because the EF PHB has the characteristics of low delay, low loss and low jitter, it is suitable for voice, video and other real-time services. Typical networks will limit EF traffic to no more than 30%, and often much less, of the link capacity.

**Assured Forwarding (AF) PHB Behaviour Group**

The AF PHB specifies a forwarding behaviour in which packets are expected to experience a very small amount of loss. The AF PHB group is a means to offer different levels of forwarding assurances for IP packets. It provides IP packets delivery in four independently forwarded AF classes (AF1, AF2, AF3, and AF4). In each DiffServ node, each AF class is allocated a certain amount of forwarding resources, and should be serviced to achieve the configured service rate. IP packets are marked with one of three
possible drop precedence values (e.g., AF11, AF12, AF13) in each class. The combination of classes and drop precedence yields twelve DSCP encodings from AF11 through AF43, as listed in Table 1.

<table>
<thead>
<tr>
<th>Assured Forwarding (AF) Behaviour Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>Low Drop</td>
</tr>
<tr>
<td>Med Drop</td>
</tr>
<tr>
<td>High Drop</td>
</tr>
</tbody>
</table>

Table 1 Assured Forwarding (AF) Behaviour Group

During congestions, the drop precedence of a packet determines the relative importance of the packet within the AF class. A congested DiffServ node protects packets carrying a lower drop precedence value from being lost by preferentially discarding packets with a higher drop precedence value. Assured forwarding allows the operator to provide assurance of delivery as long as the traffic aggregates do not exceed prescribed rates. Traffic that exceeds the prescription rate faces a higher probability of being dropped during congestions. When congestion happens between classes, the traffic in the higher class would be given priority. If congestion occurs within a class, the packets with the higher drop precedence are discarded first.

An AF implementation is expected to reduce congestion within each class. This requires an Active Queue Management algorithm. To avoid TCP global synchronization and other issues caused by Passive Queue Management (PQM), Random Early Detection (RED) or Weighted Random Early Detection (WRED) algorithms are often used to drop packets.

Class Selector PHB

Prior to DiffServ, the old version IP networks could use the Precedence field in the Type of Service (TOS) byte of the IP header to mark priority traffic. However, this field was not widely used. The DiffServ network can reuse the TOS field as DS field. In order
to maintain backward compatibility with network devices that still use the Precedence field, DiffServ defines the Class Selector PHB. The Class Selector code points are the first three bits of the IP precedence bits. Each IP precedence value can be categorized into a DiffServ class. If a packet is received from a non-DiffServ aware router that used IP precedence markings, the DiffServ router can still understand it as a Class Selector code point.

2.3 Classes

A service class represents a set of traffic requiring specific delay, loss and jitter characteristics. It is essentially a statement of the required characteristics of the traffic aggregate that can be realized by the use of associated PHB. The work of setting up the recommended configuration guidelines for DiffServ Service Classes is currently undertaken. Service classes should be defined according to the characteristics of the traffic and the required performance of the applications and services. This approach allows the mapping of current and future applications and services of similar traffic characteristics and performance requirements into the same service class.

The recommended service classes are listed as follows:

**Telephony service class:** Traffic in this class, such as IP telephony, and circuit emulation over IP applications, requires real-time transfer, very low delay variation and packet loss and is of a constant rate. They normally do not respond dynamically to packet loss. Thus AQM should not be applied. This type of service class should use EF PHB, and should also be configured to use a priority queuing system.

**Multimedia Conferencing service class:** Traffic in this class, such as video conferencing service, has specific performance requirements with respect to the exchange of continuous data like audio and video streams.

The AF PHB should be used to provide the bandwidth assurance for AF41, AF42, and AF43 marked packets to ensure their forwarding. AQM should be used primarily to switch the video encoding rate under congestion, changing from high rate to lower rate.
Multimedia Streaming service class: Traffics in this class require near-real-time packet forwarding of variable rate flexible traffic sources. The AF PHB should be used to provide a minimum bandwidth assurance for AF31, AF32, and AF33 marked packets to ensure their forwarding. The Multimedia Streaming service class traffic has the capability to react to packet loss by reducing the transmission rate. Thus, AQM should be used primarily to reduce forwarding rate to the minimum assured rate during network congestion.

Low Latency Data service class: This class is for flexible and responsive applications, such as TCP short-lived flows, data processing applications, web-based transactions, etc. The AF PHB should be used to provide a minimum bandwidth assurance for AF21, AF22, and AF23 marked packets to ensure their forwarding. As these services are flexible and respond dynamically to packet loss, AQM should be used primarily to control TCP flow rates when congestion occurs.

High Throughput Data service class: Traffic in this class requires timely packet forwarding of variable rate traffic sources, and is configured to provide good throughput for TCP long-lived flows, such as FTP. The AF PHB should be used to provide a minimum bandwidth assurance for AF11, AF12, and AF13 marked packets to ensure their forwarding in timely manner. As these services are flexible and respond dynamically to packet loss, AQM should be used primarily to control TCP flow rates when congestion occurs.

2.4 Core Congestion Control

The most popular algorithm used for DiffServ implementation is Random Early Detection (RED). The DiffServ RED provides different thresholds for different drop precedence levels. The lowest drop precedence packets will be granted the lowest minimum and maximum thresholds. Therefore they have a larger probability to be dropped or marked than packets of a higher drop precedence level.

In the case of the AF class, a RED-Based algorithm, RED with In/Out bits (RIO), is preferred. RIO uses the same mechanism as in RED, and is configured with two different
sets of parameters, one for “In” packets, and one for “Out” packets. The “In” and “Out” represents whether packets are in or out of the connection conformance agreement. There are different minimum and maximum thresholds for “In” and “Out” packets. EF packets use a separate high priority queue.

The Weighted RED (WRED) is also an algorithm extended from RED using more than one set of parameters for packets with different preferences.

Apart from RED and RED-Based algorithms, classic Proportional Integral Controller was also proposed as an AQM for DiffServ network. PI AQM achieves better performance than RED under a heavier work load and does not have the complicated parameter tuning problem that RED has experienced. However, it still has the typical PI issues. For example, it depends on network dynamic parameters, and suffers a long response time under heavy congestion. A small buffer size can also degrade its responsiveness.

All of the above AQM congestion control algorithms will be introduced in the next chapter.
Chapter 3  TCP/IP and AQM

In this chapter, the congestion control mechanism in TCP/IP environment is introduced. It consists of two main parts: the source congestion control mechanism and the router congestion control mechanism. A number of representative RED-Based AQM schemes in TCP/IP networks are studied. The scheduling and queue management schemes proposed to provide differentiated Quality of Service (QoS) regarding to the IP preferences or traffic classes are introduced as well.

3.1  TCP Congestion Control

Being a critical issue in the Internet Protocol network, congestion control has drawn a lot of attention from researchers. Although congestion is a complex process to define, and currently there is no agreed definition, the fundamental issues of congestion and different approaches to avoid and control congestion have been widely studied. Many related works can be found in the literature focusing on this area. Transmission Control Protocol (TCP) congestion control and the Active Queue Management algorithms are two main components of congestion control mechanisms. The former was a source side mechanism first developed in 1988. When congestion occurs, the TCP sender will reduce the speed of injecting data packets into the network, so that many TCP connections can share one bottleneck link. It has been proven to have played an essential role in ensuring the Internet’s robustness. Since then, many variants of TCP have been proposed to meet the Internet’s needs in the present day.

3.1.1 TCP Algorithms

There are four congestion control algorithms used in TCP and TCP-based congestion control protocols: Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery [50].

Slow Start
TCP would start a connection when the sender wants to send multiple segments into the network. The size of those segments might be up to the highest capacity of the receiver. It is no problem when the sender and the receiver are in the same network, without slow links. However, traffic transference between different LANs might be congested due to the lack of availability of queuing buffer space in some intermediate routers. Beginning transmission into a network with unknown conditions will almost certainly cause some problems. The TCP can slowly probe the network to determine the available capacity in order to avoid congesting the network with too large an amount of data at the beginning of the transfer, or after repairing loss detected by the retransmission timer by using a Slow Start. It operates by monitoring the rate at which new packets should be injected into the network is the rate at which the acknowledgments are returned by the receiver.

The Congestion Window (CWND) is a sender-side variable used to limit the number of data segment that the sender can send into the network before receiving an acknowledgement. The Receiver's advertised Window (RWND) is a receiver-side variable used to limit the amount of outstanding data. The Slow Start Threshold (SSTHRESH) is used at the sender-side to determine whether Slow Start is used to control data transmission or another algorithm, Congestion Avoidance, is.

When a new connection is established, CWND is initialized to be one segment. Each time an acknowledgement is received, CWND is increased. The sender can transmit up to the minimum of CWND and the RWND. For example, the sender transmits one segment and wait for its Acknowledgement (ACK) at the beginning of the transfer. When that ACK is received, CWND is increased from one to two. After each of these two segments is acknowledged, CWND is increased to four. The initial value of SSTHRESH might be high. Some implementations use the size of the RWND. But it may be reduced in a congestion situation. When CWND and SSTHRESH are equal to each other, Slow Start will end, and then Congestion Avoidance is used.

**Congestion Avoidance**
Congestion Avoidance uses different methods to increase CWND. It dictates that the congestion window be increased by one full sized segment every Round Trip Time (RTT). Congestion Avoidance continues to increase CWND this way until congestion is detected. Compared with Slow Start, Congestion Avoidance has a linear growth of CWND.

When congestion occurs, the SSTHRESH is set to be half of the current window size, which should be the minimum of CWND and the RWND, but at least 2 segments. Then CWND is set to one segment again. When the newly sent segment’s ACK has arrived, the congestion window will be increased in a manner, which depends on whether the TCP is in the Slow Start period or the Congestion Avoidance period.

Congestion Avoidance and Slow Start are two independent algorithms, which can be implemented together. The decision as to which of these two algorithms is used is determined by the difference between CWND and SSTHRESH. Assuming that there is no congestion, every packet will get its ACK in time, and RWND is larger than CWND, let \( c(n) \) be the value of the CWND at time \( n \), \( p(n) \) be the number of packets sent at time \( n \) and \( s \) be the value of the SSTHRESH. \( p(n-1) \) ACKs arrive at time \( n \) and the value of \( c(n) \) is therefore updated \( p(n - 1) \) times. We can then express the updating rules as shown in Figure 4:

![Figure 4](image)

If \( n = 0 \)
\[
c(n) = 1
\]

Else if \( c(n - 1) \leq s \)
\[
c(n) = c(n - 1) + p(n - 1)
\]

Else
\[
c(n) = c(n - 1) + p(n - 1) / c(n - 1)
\]

End if

**Figure 4** TCP CWND Updating Rules in Congestion Avoidance Phase

If every packet in the congestion window could be sent successfully at time \( n \), we get \( p(n)=c(n) \), then the TCP CWND is updated by the rules shown in Figure 5:
If \( n = 0 \)
\[
c(n) = 1
\]
Else if \( c(n - 1) \leq s \)
\[
c(n) = 2c(n - 1)
\]
Else
\[
c(n) = c(n - 1) + 1
\]
End if

This gives us TCP congestion window general updating rules as shown in Figure 6:

When \( c(n - 1) \leq s \), or in another word, \( n \leq N \),
where \( N \) is given by the relation \( 2N - 1 \leq s < 2N \),
\[
c(n) = 2n
\]
Otherwise
\[
c(n) = c(N) + (n - N)
\]

Figure 7 shows how the congestion window increased in different periods:
This increase in the congestion window will finally cause packet loss. When TCP detects congestion, it will set the SSTHRESH to half of the current window size and reset CWND to 1 again. Then, Slow Start starts instead of Congestion Avoidance.

**Fast Retransmit**

We know that TCP may generate an immediate duplicate ACK when an out of order segment is received. The purpose of this duplicate ACK is to let the sender know that a segment was received out of order, and to tell it what sequence number the receiver side is expecting. From the sender's point of view, duplicate ACK can be triggered by a number of problems with the Internet. One possibility is that they can be caused by missing segments. If that is the reason, all segments after the dropped segment will trigger a duplicate ACK. Here is an example of a duplicate ACK:

<table>
<thead>
<tr>
<th>Triggering-segment</th>
<th>ACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000</td>
<td>5500</td>
</tr>
<tr>
<td>5500</td>
<td>6000</td>
</tr>
<tr>
<td>6000(lost)</td>
<td>6000(duplicate)</td>
</tr>
<tr>
<td>6500</td>
<td>6000(duplicate)</td>
</tr>
<tr>
<td>7000</td>
<td>6000(duplicate)</td>
</tr>
<tr>
<td>7500</td>
<td>6000(duplicate)</td>
</tr>
<tr>
<td>6000(retransmission)</td>
<td>8000</td>
</tr>
</tbody>
</table>

Figure 8  Duplicate ACKs

Duplicate ACKs can also be caused by the re-ordering of data segments by the network or by the replication of ACKs or data segments by the network. TCP needs to wait for a small number of duplicate ACKs to be received to clarify the reason. If a duplicate ACK is just to show a reordering of a data segment, there will only be one or two duplicate ACKs before the reordered segment is processed, and then a new ACK will be generated. But if three or more duplicate ACKs are received in a row, it is a signal denoting a missing segment. TCP then performs a retransmission of the missing segment, without waiting for the retransmission timer to count down to the end.
The TCP receiver then sends an immediate ACK when the arriving segment fills in all or part of the gap in the sequence space.

**Fast Recovery**

After the Fast Retransmit algorithm sends the missing segment, the Fast Recovery algorithm controls the transmission of new data until a non-duplicate ACK arrives. Instead of Slow Start, Congestion Avoidance is performed. It is an improvement that allows high throughput under moderate congestion, especially for a large window.

The reason for performing Congestion Avoidance instead of Slow Start is that: the arrival of this duplicate ACK not only means that the receiver is waiting for a missing segment, but also means that segments are most likely leaving the network. Because the receiver generates a duplicate ACK only after a segment has arrived, the segment has now left the network and is in the receiver’s buffer. Therefore, it is no longer consuming network resources. This also means there is still data flow between these two ends, and it is not necessary to reduce the flow suddenly by going into Slow Start. The TCP can continue to transfer segments.

The Fast Retransmit and Fast Recovery algorithms are usually implemented together. The steps are shown in Figure 9.

The TCP sender only retransmits a packet after a retransmit timeout has occurred, or after three duplicate acknowledgements have arrived triggering the Fast Retransmit algorithm. A single retransmit timeout might result in the retransmission of several data packets, but each time the Fast Retransmits algorithm starts only a single data packet gets transmitted. When multiple packets have been dropped from a single window of data and the Fast Retransmit and Fast Recovery algorithms are invoked.

**3.1.2 TCP-Based Protocols**

Known as the early version of TCP, TCP Tahoe [50] enters the slow-start phase irrespective of the type of loss event. TCP Reno [51] introduces Fast Retransmit and Fast Recovery. The TCP NewReno [52] improves the Reno implementation regarding the Fast
Recovery mechanism. It aims to avoid multiple reductions in the congestion window, or unnecessary retransmit timeout with Slow Start invocation. NewReno prevents a TCP sender from reducing the congestion window multiple times while several packets are dropped from a single window of data. The NewReno remains in Fast Recovery until all of the data outstanding by the time the Fast Recovery was initiated has been acknowledged. NewReno can retransmit one lost packet per round trip time (RTT), until all the lost packets from a particular window of data have been retransmitted.

\[
\text{If } \text{dupack} = 3 \\
\quad \text{Ssthresh} = \text{maximum } \lceil \text{packets out}/2, 2 \rceil \\
\quad \text{Send the expected segment} \\
\quad \text{Set Cwnd} = \text{ssthresh} + 3 \\
\text{If } \text{dupack} = 1 \\
\quad \text{Cwnd} = \text{Cwnd} + 1 \\
\quad \text{Sent new segment if allowed} \\
\text{If new ACK (lost packet acknowledged)} \\
\quad \text{Cwnd} = \text{ssthresh} \\
\text{Else if time out} \\
\quad \text{Ssthresh} = \text{maximum } \lceil \text{packets out}/2, 2 \rceil \\
\quad \text{Cwnd} = 1 \\
\text{End if}
\]

**Figure 9** Fast Retransmit and Fast Recovery Algorithms

TCP Selective Acknowledgement (SACK) [53] is another proposed extension to TCP allowing a TCP receiver to acknowledge out-of-order packets selectively rather than just cumulatively acknowledging the last correctly received, in-order packet. Thus, TCP Sack may recover multiple lost packets in a window of data during one RTT.

TCP Vegas [54] observes the indication of congestion by changes in the RTT associated with the packets that it has sent previously, instead of the loss events. If the observed RTT increase, TCP Vegas infers incipient network congestion and reduces the congestion window by one. Otherwise, CWND is increased by one. To make TCP fully exploitable for the network capacity of fast, long-distance networks, research proposals on TCP congestion window adapting approaches include High-Speed TCP, Scalable TCP.
Chapter 3 TCP/IP and AQM

[55], FAST TCP [56] and H-TCP [57], etc. To improve the TCP Congestion Avoidance in high speed networks, the eXplicit Control Protocol (XCP) [13], and the Adaptive Congestion Protocol (ACP) [18] have been proposed in the last few years.

The TCP congestion control mechanism has proven to be very successful and necessary in avoiding congestion collapse. However, this result is based on the assumption that all of the data flows inside the network are TCP-friendly data flows, and all of them have a congestion control mechanism.

As there are more and more computers joining the Internet, computer networks have increased exponentially. Especially nowadays, when there are so many attractions on the web, and so many multimedia files which may be downloaded. The demand for high transfer speed is a growing one. Nevertheless, there are many applications that are using a non-TCP-friendly data flow, or they do not have a congestion control mechanism. And there are also some users who have initialized the TCP window to be very large and unchangeable.

Every data flow inside the network, which has one of the characteristics mentioned above, will certainly increase risk to the Internet and might become the cause of a congestion collapse.

Considering all of the aspects above, although the TCP congestion control mechanism is a very powerful tool to protect the network from being congested, the Internet cannot only rely on the TCP congestion control mechanism’s efforts to keep all data flows working smoothly in the face of a massive increase in data transfer.

3.2 AQM and PQM

The increasing demand of Internet users for time-sensitive applications with different QoS requirements has challenged the efficiency and the feasibility of the end-to-end implicit feedback based TCP paradigm. TCP-Based protocols are not sufficient to provide good service in all circumstances due to the limitation on control arising from working only from the edges of the network. This fact has led to the introduction of
Active Queue Management (AQM) mechanisms, as a form of router support to TCP congestion control. AQM mechanisms aim to provide high link utilization with low loss rate and queuing delay, while responding quickly to load changes.

Prior to AQM, Passive Queue Management (PQM) had been widely used in the Internet for many years. PQM manages the queue length inside the router by setting a maximum value for every queue in terms of packets. When the router accepts packets into the queue, the current queue length will grow until it reaches the maximum value. The router will then reject any subsequent incoming packets until the queue length has decreased. This technology is known as the Drop-Tail (DT) algorithm. Although DT is simple to implement, and has been used for many years, it was shown to interact badly with TCP congestion control mechanisms. In particular, studies have shown that DT can cause the following issues [58] [59] [60].

**Lock out**

Under some circumstances, the DT algorithm will let one data flow or several data flows occupy the queue buffer. This will prevent packets from other data flows from getting in to the queue until the queue length has been decreased. This Lock out phenomenon is usually the result of synchronization or other timing effects.

**Full queues**

The DT algorithm can only signal congestion when the queue has become full by dropping the most recently arrived packet. This will cause the queue to maintain a full or almost full status for a long period of time. However, reducing the steady-state queue size should perhaps be the most important goal for queue management algorithm, since the end-to-end delay is mainly caused by the queuing delay inside the router.

**Global synchronization**

The importance of a queue is that it can absorb data bursts and transmit them during the ensuing bursts of silence. A large buffer will certainly be able to absorb more data and increase the throughput. On the other hand, the large buffer will obviously increase
the queuing delay inside the router, since the TCP mechanism usually keeps a high queue occupation. When a significant number of TCP sources slow down at the same time due to a large amount of packet loss in a short period of time, the network resources are underutilized. An undesired cycle will then start with periods of relatively low network utilization followed by heavy congestion.

Apart from Drop-Tail, Random Drop [61] and Drop Front [62] are also two other PQM methods. The former method picks up one packet inside the queue randomly to drop when the queue is full and there is another arriving packet requiring space. The Drop Front method will drop a packet from the head of the queue in the same situation. Although both of them have successfully solved the lock out problem, the full queues problem remains.

To overcome the drawbacks of the DT scheme, dropping packets before a queue becomes full so that a source can respond to congestion before buffers overflow occurs would be one possible solution. Another approach is to control the queue length, by regulating the flow of packets from the sources. This leads to the introduction of the Active Queue Management (AQM) mechanism. The AQM algorithms, such as Random Early Detection (RED), Random Exponential Marking (REM), GREEN [63] and BLUE [64], allow the router to control when and how many packets to drop. AQM can reduce the amount of dropped packets in the router, provide lower queuing delay, and avoid the lock out behaviour.

The main AQM performance characteristics include:

**Efficient queue utilization:** The queue should avoid overflow that results in lost packets and undesired retransmissions or emptiness that results in link underutilization.

**Low Queuing Delay:** It is desirable to keep both the queuing delay and its variations small.

**Robustness:** AQM needs to maintain robust behaviour in spite of varying network conditions, such as variations in the number of TCP sessions, and variations in propagation delay and link capacity.
3.3 RED and ARED

Random Early Detection, RED is an AQM algorithm for routers that will provide advantages to Internet performance. In 1993, RED was presented in Sally Floyd and Van Jacobson's paper [20]. This algorithm detects incipient congestion by calculating the average queue length. It could notify connections for congestion either by dropping packets arriving at the gateway or by setting one bit in the packet headers to 1 or 0.

The design goals of the RED algorithm include the avoidance of global synchronization and of a bias against bursty traffic, together with the ability to maintain an upper bound on the average queue size even in the absence of cooperation from transport-layer protocols in order to avoid congestion. It can also minimize the packet dropping rate and the queuing delay.

The RED algorithm consists of two main parts. The first one is calculating the average queue length; the second is calculating the probability of dropping an incoming packet.

A. Calculating the average queue length

RED calculates the average queue length using a type of a low-pass filter with an exponential weighted moving average. This is because the packet bursts are part of the characteristics of networking. If one queue remains empty most of the time, and is fully filled in a moment, only to be empty again the next, it cannot be seen to be inception congestion.

In RED, the average queue length equals:

\[ q_{\text{ave}} = (1 - \alpha) \times q_{\text{ave}} + q_{\text{cur}} \times \alpha \]  

(3.1)

\( q_{\text{ave}} \) : average queue size, updated every packet arrival;
\( q_{\text{cur}} \) : current queue size;
\( \alpha \) : the queue weight, usually set as 0.002;
So RED is designed to respond to a time-averaged queue length, not an instantaneous one. Thus, if the queue has been empty for the most of the time, but has been just filled, RED will not drop the packet unless the queue overflows. On the other hand, if the queue has been relatively full, indicating persistent congestion, the arriving packets are more likely to be dropped.

In equation (3.1), $\alpha$ denotes the queue weight. It determines how the router reacts to changes in incoming data flows. It is very important to choose the value of $\alpha$ correctly. If $\alpha$ is too large, RED will not filter the short-lived congestion efficiently. If $\alpha$ is too small, the average queue size will not react to a change in the current queue length on time, and cannot indicate congestion. In this case inception congestion will not be detected by the RED.

$\alpha$ is pre-defined, corresponding to a different situation. In general, it depends on the size of the allowed data bursts and the persisting time in the router.

B. Calculating the probability of dropping the packet

The average queue size denotes the congestion level. After calculating the average queue length, we use its value to make the decision as to whether to drop the incoming packets or not.

There are two thresholds in RED. The $\minth$ denotes the minimum threshold while the $\maxth$ denotes the maximum threshold. When a packet reaches the router, RED will calculate the average queue size. If it is less than the $\minth$, then there is no need to drop the packet. If it is placed between $\minth$ and $\maxth$, then a probability that varies from 0 to $p_{max}$ linearly will be calculated. The packets will be dropped with this probability. If the average queue size is larger than the $\maxth$, then all of the packets will be dropped.

There is also another situation, when the queue overflows but the average queue size is smaller than the current queue length. In this case, the arriving packet will also be dropped.

Figure 10 shows how the probability changes while the average queue length varies.
Chapter 3 TCP/IP and AQM

Figure 10  RED Dropping Probability to The Average Queue Size

In RED, the dropping probability $p$ is calculated according to:

$$p_b = p_{\text{max}} \times \frac{(q_{\text{ave}} - \text{min}_{th})}{(\text{max}_{th} - \text{min}_{th})}$$

$$p = \frac{p_b}{(1 - \text{count} \times p_b)} \, (3.2)$$

$count$ : the number of incoming packets from the last dropping till the current incoming packet.

If the average queue size remains at a certain level and the $p_b$ does not change, when the number of incoming packets increases, the probability of dropping a packet can also increase.

The reason RED uses count as one extra condition to make the decision is because this can avoid global synchronization and the bias against low bandwidth data flows with high bursts.

The detailed algorithm for RED is shown in Figure 11:

When a packet arrives at an empty queue, the computation of the average queue size must take into account how much time has passed since the queue went empty. Further discussion on this method can be found in [20] [65].
Chapter 3 TCP/IP and AQM

Figure 11 RED Algorithm

The throughput and the queuing delay are mutually contradictory. The ideal average queue length can get the best results by maintaining the balance between the throughput and the delay. Usually, the length between minth and maxth should be larger than the average increment of the average queue length. Considering the characteristics of the existing data flows in the Internet, the maxth is set to be twice as big as the minth.

In the RED mechanism, when the average queue length has exceeded the threshold, the router will start to drop the incoming packets. That is to say that the packet is dropped...
while the queue is not full. In this situation, although the queue length has been efficiently managed, there is also a waste of the queue capacity. And it also affects the performance of interactive Internet applications. So, besides dropping the packet, there is another option that RED can choose, to mark the incoming packets.

ECN [66], or Explicit Congestion Notification, can be unified with RED. ECN needs a two-bit ECN in the packet IP header. One bit is the ECT, ECN-Capable Transport, indicating that the transfer protocol supports the ECN. The other is the CE, Congestion Experienced, notifying whether there is inception congestion or not. This bit is set by the router.

Two bits are also needed in the TCP header. One is ECN-Echo, the other is CWR, Congestion Window Reduced. ECN-Echo is set by the receiver notifying the sender that a CE packet has been received. The CWR is set by the sender to inform the receiver that the window size has been reduced.

Cooperation between the ECN and RED can efficiently avoid unnecessary packet dropping, and increase the utilization of the network and the throughput. It especially provides a better service to interactive applications on the Internet.

RED has brought many advantages to the Internet. However, every coin has two sides. RED parameter tuning has always been a complicated problem. The performance of RED mostly depends on the pre-defined parameters, w, minth and maxth. One group of parameters value might do well for the throughput, but from the queuing delay’s point of view, they might not be the right choice. Further explicates on how to carefully choose the values of parameters in order to keep the balance and find the right middle point is required. Moreover, tiny changes to the parameters will greatly affect the overall performance of the network. The Internet is dynamically changing all the time. The advantages that one group of parameters have brought to the network one moment might be disadvantages in the next.

Apart from that, when there are non-TCP-friendly data flows in the network along with TCP friendly data flows, the non-TCP-friendly data flows will occupy more
bandwidth than the others. The probability of dropping a packet in RED has a direct ratio with the occupied bandwidth. In this case, when the network meets the congestion, RED will get the TCP-friendly data flows into a multiple time out situation.

To solve these problems, several RED-Based algorithms have been presented, such as Stabilized RED (SRED) [31], Flow RED (FRED) [67], Dynamic RED (DRED) [68], Gentle RED [69] Adaptive RED (ARED) and New ARED [27], etc.

SRED discards packets with a load-dependent probability when a buffer in the router in the network seems to be congested. It has an additional feature that, over a wide range of load levels, it helps to stabilize its buffer occupation at a level independent of the number of active connections. SRED reaches this goal by estimating the number of active connections or flows. This estimate is obtained without collecting or analyzing state information on individual flows. The same mechanism can be used to identify flows that may be misbehaving, or are taking more than their fair share of bandwidth.

The basic idea of SRED is to compare the incoming packet to a randomly picked packet from the queue. If both of them are from the same sender, then it is classified as a hit. This method can only compare the incoming packet with a packet already inside the queue. In order to record the previous packets information, a Zombie list is designed by SRED. This Zombie List carries extra information on those data flows that have gone via the router, including the count and the time stamp.

FRED does accounting on per-active-flow basis in order to make different decision on marking the incoming packet corresponding to a different flow's bandwidth. FRED is able to successfully distinguish and limit non-TCP-friendly data flows. Hence, compared with traditional RED, it brings greater fairness into the network.

DRED uses a warning line to measure the burstiness of incoming traffic. RED with a penalty box distinguishes the TCP-friendly and non-TCP-friendly data flows when they reach the router. The non-TCP-friendly data flows will be placed into a lower class in order to protect the TCP-friendly data flows and bring fairness to RED.
RED allows network operators to simultaneously achieve high throughput and low average delay. However, the resulting average queue length is quite sensitive to the level of congestion and to RED parameter settings. It is not predictable in advance. Delay being a major problem to QoS delivered to the users, network operators would naturally like to have a rough estimate of the average delays in their congested routers; to achieve such predictable average delays with RED would require constant tuning of the parameters to adjust to current traffic conditions. Therefore, ARED sets the parameters dynamically based on the variation in the change of the average queue length. ARED solves the problem with minimal changes to the overall RED algorithm. The algorithm of ARED is as shown in Figure 12:

Every \( q_{ave} \) update:
- If \( \min_{th} \leq q_{ave} \leq \max_{th} \):
  - Status = Between;
- If \( q_{ave} < \min_{th} \) and status ≠ Below:
  - Status = Below;
  - \( p_{max} = p_{max} / \alpha \);
- If \( q_{ave} > \max_{th} \) and status ≠ Above:
  - Status = Above;
  - \( p_{max} = p_{max} \times \beta \);

Status: the status of average queue length;
- Above: larger than \( \max_{th} \);
- Between: less than \( \max_{th} \), larger than \( \min_{th} \);
- Below: less than \( \min_{th} \);
\( \alpha \): increment;
\( \beta \): decrease factor;

Figure 12 ARED Algorithm

The New ARED algorithm changes the \( p_{max} \) in every interval seconds rather than every \( q_{ave} \) update. The algorithm of New ARED is as shown in Figure 13:
Every interval seconds:
If average queue length exceeds the target and the $p_{\text{max}} \leq 0.5$:

Increase $p_{\text{max}}$:

$p_{\text{max}} = p_{\text{max}} + \alpha$;

Else if average queue length is less than target and $p_{\text{max}} \geq 0.01$:

Decrease $p_{\text{max}}$:

$p_{\text{max}} = p_{\text{max}} \times \beta$;

Interval: time; 0.5 seconds
Target: target for average queue length

$[\text{min}_{\text{th}} + 0.4 \times (\text{max}_{\text{th}} - \text{min}_{\text{th}}), \text{min}_{\text{th}} + 0.6 \times (\text{max}_{\text{th}} - \text{min}_{\text{th}})]$.

$\alpha$: increment; $\text{min} \ (0.01, p_{\text{max}} / 4)$

$\beta$: decrease factor; 0.9

Figure 13  New ARED Algorithm

ARED and New ARED can be implemented as a simple extension within RED routers, removing the sensitivity to parameters that affect RED's performance and can reliably achieve a specified average queue length target in a wide variety of traffic scenarios.

But this extension of RED has created even more parameters for traditional RED. If $\alpha$, or $\beta$ are set too large, the $p_{\text{max}}$ will change fiercely, leading to instability in the network. If $\alpha$, or $\beta$ are too small, $p_{\text{max}}$ will be changed several times to achieve the expected value. Further research is still needed on how to set the interval in the New ARED and how to initialize the $p_{\text{max}}$.

### 3.4 Weighted Fair Queuing (WFQ) and Class-Based Weighted Fair Queuing (CBWFQ)

For many years RED has guaranteed fairness among TCP connections by ensuring the fair sharing of resources and avoiding TCP global synchronization. With help from the IP header, in the recent decade, even more AQM algorithms have been proposed.
Weighted Fair Queuing (WFQ) [70], depicted in Figure 14, is one example of the algorithms developed from RED to enhance the fairness of the share.

![WFQ Structure](image)

WFQ is an automated scheduling algorithm providing fair bandwidth allocation to all network traffics. WFQ applies priorities, or weights, to identified traffics so as to classify traffics into conversations and determine how much bandwidth each conversation is allowed relative to other conversations. This flow-based algorithm schedules interactive traffic to the front of a queue to reduce response time and fairly shares the remaining bandwidth among high-bandwidth flows. In other words, by allowing the router to provide low-volumes traffic, such as Telnet sessions, priority over high-volume traffic, such as FTP sessions, WFQ gives simultaneous file transfers the balanced use of the link capacity; that is, when multiple file transfers occur, the transfers are given comparable bandwidth.

WFQ overcomes a serious limitation of First In First Out (FIFO) queuing. When FIFO is in effect, traffic is transmitted in the order received without regarding for bandwidth consumption or the associated delay requirements. Consequently, file transfers and other high-volume network applications often generate series of packets of associated data, known as packet trains. Packet trains are groups of packets that tend to move
together through the network. These packet trains can consume all available bandwidth, depriving other traffic of it. WFQ breaks up the packet trains within a conversation to ensure that bandwidth is shared fairly between individual conversations and that low-volume traffic is transferred in a timely fashion.

WFQ classifies traffic into different flows based on packet header addressing, including characteristics such as source and destination network, MAC address, protocol, source and destination port, socket numbers of the session, Frame Relay Data-Link Connection Identifier (DLCI) value, and Type of Service (ToS) value. It places packets of the various conversations in the fair queues before transmission.

In WFQ, incoming packets of high-bandwidth flows are discarded after the congestive-messages threshold has been met. However, low-bandwidth flows, which include control-message conversations, continue to enqueue data. As an IP Precedence-aware algorithm, WFQ is able to detect higher priority packets marked with precedence by the IP Forwarder, and can provide faster scheduling and superior response time for this traffic. Thus, as the precedence increases, WFQ allocates more bandwidth to the conversation during periods of congestion.

Class-Based Weighted Fair Queuing (CBWFQ) [71] is an extension of WFQ which has the functionality to support user-defined traffic classes. For CBWFQ, traffic classes are defined based on match criteria, such as protocols, Access Control Lists (ACLs), and input interfaces. The traffic for a certain class is constituted by the packet’s satisfying the criteria. A queue is reserved for each class, and the traffic belonging to each class is directed into it. The operator can then characterize a class by assigning bandwidth, weight, and a maximum packet limit.

Drop-Tail is used for CBWFQ classes unless you explicitly configure a policy for a class to use Weighted Random Early Detection (WRED) to drop packets as a means of avoiding congestion.

In standard WFQ, packets with the same source IP address, destination IP address, source TCP or User Datagram Protocol (UDP) [72] port, or destination TCP or UDP port
are classified as belonging to the same flow. WFQ allocates an equal share of bandwidth to each flow. Flow-based WFQ is also called fair queuing because all flows are equally weighted.

For CBWFQ, the weight specified for the class turns into the weight of each packet satisfying the class match criteria. After a packet is assigned with weight, it is enqueued in the suitable class queue. CBWFQ uses the weights assigned to the queued packets to ensure that the class queue is serviced fairly.

### 3.5 WRED and RIO

The Internet nowadays is expected to provide distinct service to different users based on value, and to deliver distinct service based on content accessed, etc. The WFQ is a weight-related scheduling algorithm implemented in congestion control routers. Queue management also plays another important role in the router. As with scheduling, queue management is expected to provide a distinct treatment to the traffic based on the weights of the flows. However, the fairness-addressed characteristic makes QoS differentiation impossible for traditional RED, or many other RED-based algorithms. Thus, priority related RED algorithms, such as Weighted RED (WRED) and RED with In/Out bits (RIO), are proposed to provide early detection with some QoS considerations.

Both WRED and RIO are extensions to RED. WRED classifies packets according to their IP precedence and Type of Services (ToS) field in the earlier version IPv4 header, or the Differentiated Services Code Point (DSCP) in the Differentiated Services (DS) field in the current IPv4 and IPv6 header. Depending on certain parameters, such as source address, destination address or traffic type, the packets can be assigned to specific traffic classes by the classifiers, and their IP precedences and DS fields will be marked correspondingly. WRED then provides separate RED parameters sets, such as thresholds and weights, so as to allow the network to provide different Qualities of Service in regard to packet dropping for different traffic types. The structure of WRED is shown in Figure 15.
Chapter 3 TCP/IP and AQM

WRED differs from other Congestion Avoidance techniques, such as queuing strategies, because it attempts to anticipate and avoid congestion before it happens, rather than control congestion once it occurs. WRED makes early detection of congestion possible and provides for multiple classes of traffic. However, WRED is usually used in the core routers of a network, rather than at the network boundary. Edge routers assign IP Precedences to packets as they enter the network. WRED uses these precedences to determine the appropriate treatment.

WRED is only useful when the bulk of the traffic is TCP/IP traffic. The non-IP traffic is treated as precedence 0, the lowest precedence. Therefore, it is more likely to be dropped than IP traffic.

RIO, on the other hand, has a Time Sliding Window (TSW) tagger profile meter designed for bulk-data transfer. TSW maintains three state variables: Win-length, which is measured in units of time, Avg-time, the rate estimate upon each packet arrival, and T-front, which is the time of the last packets arrival. Since TSW is used to estimate the rate upon each packet arrival, state variables Avg-rate and T-front are updated each time a packet arrives. However, Win-length is pre-configured. The TSW algorithm and the marking algorithm are shown in Figure 16 and Figure 17.
Initially:

Win-length = a constant;
Avg-rate = flow's minimum guaranteed rate;
T-front = 0;

Upon each packet arrival, TSW updates its state variables as follows:

Bytes-in-TSW = Avg-rate * Win-length;
New-bytes = Bytes-in-TSW + pkt-size;
Avg-rate = New-bytes /(now - T-front + Win-length);
T-front = now;

now: time of current packet arrival
pkt-size: packet size of the arriving packet

Figure 16  RIO TSW Algorithm

If (Avg-rate <= flow's minimum guaranteed rate) then
the arriving packet is tagged as in packet:
else
P = (Avg-rate - flow's minimum guaranteed rate) /Avg-rate;
if (P = rand(0,1)) then
the arriving packet is tagged as out packet:
else
the arriving packet is tagged as in packet;

rand(0,1): random number in the range from 0.0 to 1.0

Figure 17  RIO Marking Algorithm

The tagger estimates the TCP sending rate smoothly upon each packet arrival and tags packets as OUT when the traffic exceeds a certain threshold. If the packet is an IN packet, the router calculates avg_in, the average queue for the IN packets; if it is an OUT packet, the router calculates avg_total, the average total queue size of all (both IN and OUT) arriving packets. The probability of dropping an IN packet depends on avg_in and the probability of dropping an OUT packet depends on avg_total.

RIO drops OUT packets more aggressively than IN packets. RIO can achieve fair bandwidth allocation in the network only if congestion sensitive flows exist. However, if congestion sensitive and insensitive flows co-exist in the network, then RIO cannot achieve fair bandwidth allocations. The structure of RIO is shown in Figure 18.
In WRED, different queues may be configured with separate sets of RED parameters or with different buffer occupation thresholds, and packets are classified into these queues according to the priority information assigned to the packets before they are transmitted on the further hop. Eight separate levels of drop precedence can be supported. RIO, on the other hand, has only two sets of RED parameters and the same queue is shared by both IN packets and OUT packets. RIO discriminates against OUT packets in times of congestion. Since parameterization has always been a complicated part of RED algorithm, WRED and RIO have, to some extent, eroded this problem, as both of them are using more than one set of parameters to manage the queue. Furthermore, WRED uses a set of queues to store incoming packets based on their priority information. When the average queue length of one queue is under its thresholds, it may still not be able to accept the incoming packets because they are carrying unmatched priority information. This leads to a probability that the buffer utilization is inferior throughout the whole transmission.

3.6 PI

As another AQM algorithm developed from the classical control system techniques, the Proportional Integral (PI) [73] controller is said to be a natural choice due to its robustness and its ability to eliminate the steady-state error. Although it is said that PI would be difficult to implement, compared to RED, PI still has some advantages, such as it does not need very complicated parameterization, and provides the ability to efficiently control queue size under heavy work load. PI attempts to maintain an explicit target
queue length $q_{\text{ref}}$. It samples instantaneous queue length at fixed intervals and computes a dropping probability $p_p$, for the incoming packet.

A simplified version of a dynamic TCP model using fluid-flow and stochastic differential equation analysis is used and linearized in the PI controller. The TCP/AQM Block diagram along with a PI control law implementation with the TCP/AQM dynamic is shown in Figure 19. It ignores both the Slow Start phase and the timeout mechanism of TCP.

$C(s)$ is the transfer function of AQM controller. $P(s)$ is the transfer function of TCP/AQM plant. $R_0$ represents the time delay. $\delta p$ and $\delta q$ denote the fluctuation of dropping probability and the fluctuation of queue size, respectively.

Using this TCP/AQM dynamics, the PI controller is expected not only to improve the responsiveness of the TCP/AQM dynamics but also as to stabilize the router queue length around the desired value $q_0$. The latter can be achieved by means of integral control, while the former can be achieved by means of proportional control using instantaneous queue length instead of using the average queue length.

![Figure 19 PI AQM Block Diagram](image)

The procedure of implementation in the AQM enabled router is shown in Figure 20.
Upon receiving one packet:
\[
P_{pi}(kT) = a_{pi} \cdot (q(kT) - q_{ref}) - b_{pi} \cdot (q((k-1)T) - q_{ref}) + p_{pi}((k-1)T)
\]

If \( P_{pi}(kT) < 0 \) then \( P_{pi}(kT) = 0 \);
If \( P_{pi}(kT) > 1 \) then \( P_{pi}(kT) = 1 \);
Drop the packet with probability \( P_{pi} \);

- \( a_{pi}, b_{pi} \) and \( T \) depend on link capacity, maximum RTT and the number of flows at a router.
- \( a_{pi}, b_{pi} \) are the PI controller parameters.
- \( T \) is the sampling time.
- \( k \) is the sample instant.

Figure 20. PI AQM Algorithm

However, the properties of this controller are not desirable, since it depends on network dynamic parameters, such as the number of flows and the round-trip time. The research shows that the response time of PI is dependent on the control constant parameters: buffer size, desired queue length, and desired stable packet drop probability, which is an increasing function of the number of TCP flows and a decreasing function of round-trip time and link capacity [74] [75]. Under heavy congestion, PI suffers from a long response time. In addition, with a small buffer size, the responsiveness of PI will become worse as well.

3.7 Conclusion

Quality of Service (QoS) refers to the capability of a network to provide better service to selected network traffic over various technologies, including Frame Relay, Asynchronous Transfer Mode (ATM), Ethernet and 802.1 networks, SONET, and IP-routed networks that may use any or all of these underlying technologies. The primary goal of QoS is to provide priorities including dedicated bandwidth, controlled jitter and latency (required by some real-time and interactive traffic), and improved loss characteristics. Also important is making sure that providing priorities for one or more flows does not cause other flows to fail. QoS technologies provide the elemental building blocks that will be used for future business applications in campus, WAN and service provider networks.
Working on the cooperation between AQM and TCP is the main approach to solve the Internet congestion control and to improve the QoS delivered to the end users. There are many theoretical analysis of TCP/RED system in framework of feedback control theory based on the continuous-time model [76] [77] [78] [79] or discrete-time model [80] [81] after having made some necessary simplification and assumption, and finally provided some very revelatory and significant conclusions and judgments.

Fundamentally, QoS enables the network to provide better service to particular data flows. This goal can be achieved either by raising QoS for a certain data flow, or by limiting the service provided to another data flow. When using router congestion management algorithms, such as RED, the QoS for a flow can be raised by queuing and servicing queues in different ways. RED can be modified to raise quality by dropping packets from lower-priority flows before packets from higher-priority flows.

All of the suggestions above are based on the assumption that the router can be informed as to what level of QoS a particular data flow needs from the packets from that data flow. In this case, the packet should carry extra bit information to denote the service level demanded. In another word, those packets should inform the router of their different priorities.

From Chapter 4 to Chapter 7, we have presented a list of AQM algorithms, which are developed to sense the user QoS Requirement and provide differentiated QoS according to user demand. Compared to traditional AQM algorithms, our algorithms can provide much more meticulous QoS consideration during transmission in TCP/IPv6 networks, protect the throughput of both Non-Priority and High-Priority dataflows under heavy congestions by using easily adoptable approaches.
Chapter 4 AQM Algorithms with Priority Checking Mechanism for TCP/IPv6

In this chapter, our first two PC-AQM algorithms, QoS Requirement aware RED (QoSR-RED) and QoS Alert aware RED (QoSA-RED) are presented. QoSR-RED is inherited from traditional RED. It provides different dropping probabilities according to QoS requirements of dataflows. QoSA-RED provides different dropping probabilities according to QoS Requirements of dataflows and their current QoS status.

The IPv4 header has been acting as an assisting component to differentiated QoS based AQM algorithms for a very long time. However, the growth of the Internet has urged the Internet Engineering Task Force (IETF) to produce an upgraded Internet Protocol IPv6 to satisfy the demands for IP addresses. The 20 bits flow label field, which is a new field added in the IPv6 header, is used to facilitate the identification of data requiring special handling, such as those involved in real-time applications, etc. Having support from the new flow label field, Active Queue Management algorithms, such as RED, with a Priority Checking Mechanism are developed to ensure the end user with high priority gets the level of QoS they expected. PC-AQMs implements a Priority Checking function in AQM, in which the flow label field is used to label the packets of different dataflow with their priority and the current level of service they have experienced, in order to request from the network router with PC-AQM distinct QoS. Served by this mechanism, real-time system users or people who would like to pay more for better services can get the guaranteed high level of QoS according to their demands.

The QoSR-RED has a priority checking function to check the QoS Requirement, labelled in the IP header, and to notify the queue management function to perform distinctly different behaviours. The QoSA-RED is an improved version of QoSR-RED, in which the extra flow label field is used to label the packets of different dataflow with their priority and the current level of service they are experiencing in order to request from the network router a distinct QoS, so that real-time system users or people who would like to pay more for better services can get the guaranteed high level of QoS.
demanded. Both QoSR-RED and QoSA-RED overcome the under-utilization of the queue and the complications of parameterization by using one buffer for multi-dataflows, one set of parameters and a different mechanism to choose which packet to drop during congestion, with the support of the IPv6 flow labelling ability.

4.1 RED with QoS Requirement Checking (QoSR-RED)

4.1.1 QoSR-RED algorithms

The QoS Requirement can be labelled in the packet header in an IPv6 network. In the PC-AQM router, an extra database is set up referred to as the Priority List, which contains the QoS information carried by the packet header along with the data generated by the router itself. Compared to traditional AQM algorithms, the structure of PC-AQM has the extra functionality of checking and labelling the IPv6 headers with QoS related information based on the Priority List, which is taken into account by the discard tester during congestion. The generic structure of PC-AQM algorithms is shown in Figure 21.

![PC-AQM Generic Structure](image)

The algorithm of QoSR-RED \[82][83] is rather simple. The priority list carries flow identification, corresponding QoS Requirement and the update status as shown in Table 2. It is initially empty. QoSR-RED updates this list when packets reach the interface. A count down timer is used to trigger the Priority List Cleaner function. If one entry of the list has not been updated during the last count down timer interval, QoSR-RED would consider this dataflow dead and delete the entry. The average queue length is calculated upon every packet arrival. The incoming packet will be dropped by the dropping probability generated from the average queue length using the traditional RED probability calculation function, if its QoS Requirement is less than or equal to the lowest
Chapter 4 AQM Algorithms with Priority Checking Mechanism for TCP/IPv6

QoS Requirement value in the Priority List. Otherwise, the QoSR-RED would pick the packet carrying the lowest QoS requirement value in the queue to drop.

|------------|---------------------|------------------|

Table 2 QoSR-RED Priority List Entry Structure

More specifically, each entry in Priority List consists of the following three variables.

1. Flow ID ($f_{id}$): Unique identification of every dataflow.

2. QoS Requirement ($q_r$): Current QoS requirement defined by the end user.

3. Update Status ($sup$): 1 for having been updated during the last count down timer interval, or 0 for dead dataflow.

The tagging algorithm at each packet arrival is shown in Figure 22, and the buffer management and packet dropping algorithm is shown in Figure 23.

Upon each packet arrival, Priority list updates its state variables as follows:
- Check whether it carries a new Flow ID;
- If Flow ID is new, then
  - Insert a new entry to the list; $f_{id} = \text{Flow ID}$;
  - Get the QoS requirement from the QoSR field; $q_r = \text{QoSR}$;
  - $sup = 1$;
- Else
  - $q_r = \text{QoSR}$;
  - $sup = 1$;
- If $q_r \leq$ the lowest QoSR value in the queue
  - Tag the packet to be droppable;

Upon every count-down timer interval, Priority list updates its state variables as follows:
- If ($sup = 0$) then
  - Delete the entry;
  - Set all of the $sup$ field in the priority list to be 0;

Figure 22 QoSR-RED Tagging Algorithm
Upon receiving one packet:
- Calculate the average queue size;
- If (average queue size > Max threshold) then
  Drop the packet;
- Else if (average queue size > Min threshold) then
  Calculate $P_{\text{red}}$, the dropping probability as traditional RED does;
  
  If $Q_r$ <= the lowest QoS value in the queue
  Drop the packet with probability $P_{\text{red}}$;
  
  Else
  Drop the droppable packet in the queue with the probability $P_{\text{red}}$; Or the last packet in the queue with the probability $P_{\text{red}}$ if there is no droppable packet;
- Else
  Accept the packet;

Figure 23 QoSR-RED Buffer Management and Packet Dropping Algorithm

4.1.2 QoSR-RED NS Simulation

Simulations are carried out in Network Simulator 2 [84] to verify QoSR-RED’s ability to provide differentiated QoS according to the QoS Requirement information carried in the packets’ IPv6 headers. Figure 24 shows a single bottle-link network with AQM algorithms implemented in ROUTER 1.

Simulation 1
In the first simulation, we look at a network with 60 FTP flows. The network also has 180 HTTP sessions, which are used here as the noise in the system. The link bandwidth is 15 Mb/s, and the propagation delays for the flows range uniformly between 80 and 120 ms, with average packet size being 500 Bytes. The buffer of the queue is 800 packets. For both traditional RED and QoSR-RED, we use the parameters derived for stable operations in [21], with maximum dropping probability being 0.1, and minimum and maximum queue size thresholds being 150 and 700 respectively. The average weight used to calculate the average queue size is set to be 1.33e-6.

For QoS Requirements assigned to every dataflow, we use a set of numbers, from 0 to 3, indicating levels of QoS Requirements from nothing to the maximum, as the QoSR-RED only compares the relative values of the QoS Requirements while choosing the droppable packet, but does not use the dropping action to achieve the precise value of the QoS Requirement. 60 FTP dataflows are equally divided into 4 groups and assigned with these four levels of QoS Requirement respectively.

During t=0-150s, 40 FTP flows are active, which cause slight congestion on the link. At t=150s and t=400s, 10 more FTP flows are added to the network respectively to further shift the level of congestion. The total duration of this simulation is 600 seconds.

The queue length plots of RED and QoSR-RED are shown in Figure 25. The dark line represents instant queue size, while the light line denotes the average queue size.

![Queue Length of RED and QoSR-RED in Simulation 1](image)
When there are only 40 FTP flows in the network, the queue size of both RED and QoSR-RED remain around the desired minimum threshold 150. At the 150th second, both of them go up to nearly 600 packets until the average queue size rises smoothly to the minimum threshold and triggers the packet dropping function. It takes around 80 seconds for the RED to stabilize the queue size back to around 150 packets, and around 90 seconds for the QoSR-RED. During t = 250 ~ 400s, the oscillation of the queue size in QoSR-RED router is slightly larger than in RED router. However, it is still acceptable. When another 10 active FTP flows start to inject their packets into the network, both RED and QoSR-RED respond rapidly. The average queue sizes in both cases rise just a little. However, this time it seems that RED has a queue which oscillates slightly more than QoSR-RED.

The slower response in QoSR-RED during t = 150 ~ 240s could be a result of the priority list maintenance function. In a small scale network, the effect brought by this function may not be that obvious. However in a network with a much heavier work load, it would take longer to build up and update the entire list every count down timer interval.

The throughputs of one FTP flow from every group respectively under RED and QoSR-RED are shown in Figure 26 and Figure 27.

Since RED does not respond to the QoS Requirement carried by the packet header, only the throughput of flow 0 and flow 3 are presented. It is shown that no matter the level of congestion in the network, both flow 0 and flow 3 have the same level of throughput.

However, when the router is implemented with QoSR-RED, the situation changes. At stage 1, when there are 40 FTP flows, the levels of throughput already indicate differences. Flows with no QoS Requirement have a throughput of around 40 Kbytes/s. The other flows share the same level of throughput, from 47.5 Kbytes/s to the maximum 50 Kbytes/s. When the queue gets congested while the work load increases to 50 FTP flows, the difference becomes more obvious. The throughput of Flow 0 reduces into only half of its original value. Throughputs of Flow 1 and Flow 2 also reduce slightly. But the
average throughput of Flow 3 does not really change. When the number of FTP flows rises up to 60, Flow 0 has a throughput less than 10 Kbytes/s, while the other flows have an 3 times larger throughputs. Flow 3 has, in average, a 32.5 Kbytes/s throughput under RED and 40 Kbytes/s under QoS-R-RED, when the congestion gets serious.

Figure 26 Throughput of FTP Flows under RED during Simulation 1

Figure 27 Throughput of FTP Flows under QoS-R-RED during Simulation 1
The simulation shows that QoSR-RED has provided the dataflows with different dropping priorities so as to efficiently improve the QoS based on the QoS Requirements they carry.

As is evident from these plots, the QoSR-RED guarantees FTP flows with high QoS Requirements a much better throughput under heavy congestion compared to FTP flows with low or even no QoS Requirements. This clearly demonstrates the ability of the QoSR-RED in assuring QoS for real-time or urgent information transmission if it has a higher priority. However, it has been noticed that FTP flows without a QoS Requirement have been scarified for the better service of the other FTP flows.

To relieve such phenomena, we take the QoS status of the flow into account in designing QoSA-RED.

4.2 RED with QoS Alert Checking (QoSA-RED)

Packets need to be labelled with QoS Requirements with precise value, indicating the desired dropping limit to support QoSA-RED [85][86]. One extra bit in the IPv6 header, which is referred to as QoSA (Quality of Service Alert), is also used to indicate whether the dataflow has been treated worse than was expected. In QoSA-RED, the priority list contains the current status of every data flow passing through the router, as shown in Table 3. The tagger will then mark the flow label field with regards to the information that the list carries.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Drop Counter</td>
<td>Arrival Counter</td>
</tr>
</tbody>
</table>

Table 3 QoSA-RED Priority List Entry Structure

Each entry in Priority List consists of the following variables.

1. Flow Id (f_id): Unique identification of every dataflow.
2. QoS Requirement \((q_t)\): Current QoS requirement defined by the end user.

3. Average Dropping Rate \((a_d)\): Weighted Moving Average of the ratio of the amount of dropped packets to the amount of arrived packet of the same flow during every time interval.

   3.1 Dropped Counter \((n_d)\): The amount of dropped packets.

   3.2 Arrived Counter \((n_a)\): The amount of arrived packets.

   3.3 Weight Factor \((w)\): Weight factor that is used to calculate the Average Dropping Rate.

4. Current Treatment \((q_a)\): Current QoS status that the data flow is being treated in.

4.2.1 Tagging algorithm at packet arrival

QoSA-RED routers do not have the authority to alter \(q_t\), but can mark the Quality of Services Alert (QoSA) field in the header with the current QoS status, so as to provide necessary references for further actions. The tagging algorithm upon each packet arrival is shown in Figure 28.

The end user can dynamically change the QoS Requirements depending on the amount he or she is willing to pay for the network services. For example, a consumer may prefer to pay more for the Internet during office hours, or pay less while downloading less important files. The QoSA-RED will notice this change once the packet with the new QoS arrives.

4.2.2 Buffer management and packet dropping algorithms in QoSA-RED

After tagging the incoming packet, the QoSA-RED router will then monitor buffer occupancy and determine whether this received packet should be dropped or not. The pseudo-code is shown in Figure 29.
Upon each packet arrival, Priority list updates its state variables as follows:

Check whether it carries a new Flow ID;

If Flow ID is new, then

- Insert a new entry to the list; \( f_{id} = \text{Flow ID} \);
- Get the QoS requirement from the QoSR field; \( q_r = \text{QoSR} \);
- \( a_d = 0; \ n_d = 0; \ n_a = \text{new packet size} \);

Else

- \( n_a = n_a + \text{new packet size} \) (where \( f_{id} = \text{Flow ID} \);
- \( q_r = \text{QoSR} \);
- If \( q_a = 0 \) then
  - Tag the packet to be droppable;

Upon every count-down timer interval, Priority list updates its state variables as follows:

- \( a_d = a_d \times (1 - w) + w \times (n_d / n_a) \);
- \( n_d = 0; \ n_a = 0 \);
- Compare Average Dropping Rate with the labeled QoS Requirement;
  - If \( (a_d \geq q_r) \) then
    - \( q_a = 1 \);
  - Else
    - \( q_a = 0 \);

Figure 28 QoSA-RED Tagging Algorithm

Upon receiving one packet:

- If (average queue size > Max threshold) then
  - Drop the packet;
- Else if (average queue size > Min threshold) then
  - Calculate \( P_{red} \), the dropping probability as traditional RED does;
  - Drop the packet with probability \( P_{red} \) if it is droppable;
- Else
  - Drop the droppable packet in the queue with the probability \( P_{red} \);
  - Or the last packet in the queue with the probability \( P_{red} \) if there is no droppable packet;
  - Else
    - Accept the packet;

Upon dropping one packet, Priority list updates its state variables as follows:

- \( n_d = n_d + \text{dropped packet size} \) (where \( f_{id} = \text{Flow ID} \);

Figure 29 QoSA-RED Queue Management and Dropping Algorithm

When the average queue size is between the two thresholds, the dropping probability of the received packet is inversely proportional to its QoSA value. Thus, a packet carrying a 0 QoSA value would be more likely to be dropped when compared to a packet with a 1 QoSA value.
This mechanism allows the router to use only one set of RED parameters to calculate the overall dropping probability $P_{\text{red}}$ from the average queue length. However, when breaking the incoming dataflows into different categories based on the tagged QoSA value, $P_{\text{red}}$ can be divided into differentiated dropping probabilities. Thus QoSA-RED can provide different levels of QoS to dataflows with different QoSA by manipulating their dropping probabilities.

4.3 QoSA-RED Modeling and Parameter Setting

4.3.1 QoSA-RED Modeling

A dynamic model of TCP behaviour was developed using fluid-flow and stochastic differential equation analysis in [78]. Based on that, a simplified TCP-queue dynamic model is also presented.

\[
\begin{aligned}
\dot{W}(t) &= \frac{1}{R(t)} - \frac{W(t) \times W(t-R(t))}{2R(t)} \times p(t-R(t)) \\
\dot{q}(t) &= \frac{W(t)}{R(t)} \times N(t) - C
\end{aligned}
\]

(4.1)

Where $\dot{x}$ denotes the time-derivative of $x$, written as

\[
\dot{x}(t) = \frac{\partial}{\partial t} x(t), \text{ and}
\]

$W =$ expected TCP window size in packets;

$q =$ expected queue length in packets;

$R =$ round-trip time $= \frac{q}{C} + T_p$ (secs);

$C =$ link capacity in packets per second;

$T_p =$ propagation delay in seconds;
$N = \text{load factor, indicating the number of TCP sessions in total;}$

$p = \text{the probability of packet dropping;}$

When taking $p$ as the system input and $(W, q)$ as the output, the steady-state operating point $(W_0, q_0, p_0)$ is defined by:

$$W_0 = \frac{1}{R_0} - \frac{W_0 \times W_0}{2R_0} \times p_0 = 0$$

$$q_0 = \frac{W_0}{R_0} N_0 - C = 0$$

Assuming the load factor $N$ and the round-trip time $R$ are two constants throughout the transmission, we can get the following two equations.

$$\begin{cases}
W_0^2 p_0 = 2 \\
W_0 = \frac{RC}{N}
\end{cases} \quad (4.2)$$

Based on equations (4.1) and (4.2), [21] has presented a simplified TCP-queue dynamic model, in which the $\delta p$ is the system input, $(\delta W, \delta q)$ is the output state. The simplified TCP-queue Dynamics is shown in Figure 30.

Where

$$\delta W = W - W_0$$

$$\delta q = q - q_0$$

$$\delta p = p - p_0$$
The original block diagram of TCP/AQM with RED controller is shown in Figure 31:

![TCP/AQM Block Diagram](image)

Where

\[ C_{red}(s) = \frac{R_0 C^2}{s + K}, \quad P_{tcp}(s) = \frac{2N^2}{s + 2N}, \quad P_{queue}(s) = \frac{N}{s R_0^2}, \quad L_{red} = \frac{p_{max}}{\text{max}_t h - \text{min}_t h}, \quad K = \frac{\log_e (1 - \alpha)}{\delta} \]

\( \alpha \) is the weight factor used to calculate the average queue size, and \( \delta \) is the sampling time, which equals to \((1/C)\). \( C_{red}(s) \) is the transfer function of RED controller. \( P_{tcp}(s) \) and \( P_{queue}(s) \) are the transfer function of TCP and Queue dynamics, respectively. When introducing Priority Checking Mechanism into the traditional RED controller, the High-Priority flows and the Non-Priority flows are actually treated differently. As the latter experiences a larger probability to be with small throughput and high dropping probabilities during congestion when the number of High-Priority flow increases in the network, we can get the block diagram of TCP/PC-RED as shown in Figure 32.
Chapter 4 AQM Algorithms with Priority Checking Mechanism for TCP/IPv6

Where

\[ K_n = 1 - K_p \]

\[ K_p \in \left[ 0, \frac{R_p}{q} \right] \] when all High-Priority flows are holding 1 as QoS Alert.

\[ r_p \] is the ratio of High-Priority flow to the overall load level N.

\[ P_n(s) = \frac{\frac{R_0 C^2}{2N_n^2}}{s + \frac{2N_n}{R_0^2 C}} \frac{N_n}{R_0} = \frac{\frac{C^2}{2N_n}}{s + \frac{1}{R_0}} \left( s + \frac{2N_n}{R_0^2 C} \right) \]

\[ P_p(s) = \frac{\frac{R_0 C^2}{2N_p^2}}{s + \frac{2N_p}{R_0^2 C}} \frac{N_p}{R_0} = \frac{\frac{C^2}{2N_p}}{s + \frac{1}{R_0}} \left( s + \frac{2N_p}{R_0^2 C} \right) \]

Let

\[ P_n'(s) = K_n P_n(s), \quad P_p'(s) = K_p P_p(s) \]

We have the following block diagram as shown in Figure 33:

Figure 33 TCP/PC-RED Block Diagram with \( P_n(s) \) as Control Plant

Thus, the simplified TCP/PC-RED block diagram is shown in Figure 34, when we consider \( P_n(s) \) as the control plant of PC-RED controller.

Figure 34 Simplified Block Diagram of PC-RED with TCP Sources

61
Where

\[
C_{pc-red}(s) = \frac{1 - K_p \left( \frac{2r_p N}{R_0^2 C} \right) + K_p \left( \frac{2(1 - r_p) N}{R_0^2 C} \right)}{1 - r_p \left( \frac{2r_p N}{R_0^2 C} \right)} \cdot C_{red}(s)
\]

\[
P_p(s) = \frac{C^2}{\left( s + \frac{2 \cdot (1 - r_p) \cdot N}{R_0^2 C} \right) \left( s + \frac{1}{R_0} \right)}
\]

### 4.3.2 QoSA-RED PARAMETER SETTING

Apart from the traditional RED parameters, such as \( \text{max}_\text{th}, \text{min}_\text{th}, \alpha \) and \( p_{\text{max}} \), which have been introduced earlier, the extra pre-defined parameters in QoSA-RED are \( r_{p_{\text{max}}} \) and \( w \). \( r_{p_{\text{max}}} \) is the maximum threshold of \( r_p \), the ratio of the high-priority dataflows to the overall data load \( N \), above which the QoSA-RED has difficulty in guaranteeing high throughput to high-priority dataflows. \( w \) is the weight factor which is used to calculate the average dropping rate \( a_d \) and can be tuned to adjust the responding speed of \( a_d \) to the dropping action.

**A. \( r_{p_{\text{max}}} \) and System Stability**

To check the system stability, Nyquist Criterion is a very popular method. Considering our system to be a Time Delay System, we need to first check whether Nyquist Criterion is applicable based on the method presented in [87].

Under heavy congestion, assuming all High-Priority flows have sensed average dropping rate exceeding QoS Requirements. Thus, we have \( K_p \approx 0 \). Let \( G(s) \) be the open loop transfer function of TCP/PC-RED.
Chapter 4 AQM Algorithms with Priority Checking Mechanism for TCP/IPv6

It is easy to observe that the degree of \( N(s) \) is less than the degree of \( O(s) \). Thus, for any \( G(s) \) with the leading coefficient of \( N(s) \) larger than the leading coefficient of \( O(s) \), Nyquist Stability Criterion is applicable. So the relationship between \( N \) and \( C \) should follow:

\[
G(s) = \frac{1 - K_p \left( s + \frac{2r_p N}{R_0^2 C} \right) + K_p \left( s + \frac{2(1-r_p)N}{R_0^2 C} \right)}{1 - r_p \left( s + \frac{2r_p N}{R_0^2 C} \right)} \cdot \frac{L_{red} \cdot K \cdot \frac{C^2}{2N_n}}{s + \frac{2N_n}{R_0^2 C} \left( s + \frac{1}{R_0} \right)}
\]

\[
= \frac{L_{red} \cdot K \cdot \frac{C^2}{2N_n}}{(s + K) \left( s + \frac{2N_n}{R_0^2 C} \right) \left( s + \frac{1}{R_0} \right)} = \frac{L_{red} \cdot K \cdot \frac{C^2}{2N(1-r_p)}}{(s + K) \left( s + \frac{2N(1-r_p)}{R_0^2 C} \right) \left( s + \frac{1}{R_0} \right)} = \frac{N(s)}{D(s)}
\]

It is easy to observe that the degree of \( N(s) \) is less than the degree of \( D(s) \). Thus, for any \( G(s) \) with the leading coefficient of \( N(s) \) larger than the leading coefficient of \( D(s) \), Nyquist Stability Criterion is applicable.

So the relationship between \( N \) and \( C \) should follow:

\[
N < \frac{p_{max} \cdot \alpha \cdot C^3}{2(\max_{\alpha} - \min_{\alpha})(1-r_p)}
\]

\( N \) is the network load level, \( r_p \) denotes the ratio of High-Priority dataflow, \( N_p \), to \( N \). \( p_{max} \) is the maximum dropping probability. \( C \) is the link capacity. \( \alpha \), \( \max_{\alpha} \) and \( \min_{\alpha} \) are RED parameters, which represent the weight factor used to calculate average queue size, the maximum and minimum thresholds, respectively.

When we set up a network with a bottle link with 125 packets per second as capacity, based on the stability requirement presented in [21] for RED controllers, if we use a PCRED router with 150 packets Buffer, minimum and maximum thresholds are 50 and 150 packets respectively, and the weight factor is \( 5e-4 \), \( r_p \) is 30%, we can get the region of \( N \) to use Nyquist Criterion in C-N plane as shown in Figure 35.
The PC-RED needs to hold a short Priority List so as to keep the delay caused by the maintenance as small as possible. In this case, it is not necessary to set up a network with more than 300 dataflows, and the region of $N$ to use Nyquist Criterion in C-N plan is shown in Figure 36.
\[ K = \frac{-\log_e (1 - \alpha)}{\delta} \]

When \( \alpha \ll 1 \), we can obtain

\[ \alpha = 1 - e^{-K\delta} \approx 1 - (1 - K\delta) = K\delta \]

Thus

\[ K \approx \frac{\alpha}{\delta} = \alpha C \]

When we choose \( N = 20 \), \( R = 0.2 \), \( G(s) \) is a transfer function with:

\[ G(s) = \frac{k_0}{(s - p_1)(s - p_2)(s - p_3)} \quad (4.4) \]

With

\[ k_0 = \frac{p_{\text{max}} \cdot \alpha \cdot C^2}{2N \cdot (1 - r_p)(\text{max}_{\text{th}} - \text{min}_{\text{th}}) (1 - r_p)} = \frac{0.05}{(1 - r_p)} \]

\[ p_1 = -\alpha C = -0.0625; \]

\[ p_2 = -\frac{2(1 - r_p)N}{R_0^2 C} = -8(1 - r_p); \]

\[ p_3 = -\frac{1}{R_0} = -5. \]

The Nyquist Plot of \( G(s) \) is shown in Figure 37 with various \( r_p \).
Chapter 4 AQM Algorithms with Priority Checking Mechanism for TCP/IPv6

Figure 37 Nyquist Plot of G(s) with various \( r_p \)

We choose two critical settings to analyze the system stability by using a Bode Plot. Assume the total amount of High-Priority flow only forms a small portion of the overall work load, the probability for the QoS to the Non-Priority flow to be undermined is rather small. Thus we choose \( r_p \) to be 0.005. The Bode Plot of the system is shown in Figure 38. It is observed that the system has a negative gain margin and the Phase is larger than -180 degree when the Magnitude is 0dB.

Figure 38 Bode Diagram of G(s) with 0.005 as \( r_p \)
When the network is fulfilled with High-Priority flow, $r_p$ will press on towards 100%. The Bode Plot of $G(s)$ when $r_p$ is 0.995 is shown in Figure 39. The Gain Margin is also negative while the Magnitude has never reached 0dB.

![Bode Diagram](image)

Figure 39  Bode Diagram of $G(s)$ with 0.995 as $r_p$

We can reach the conclusion that if the PC-RED parameters satisfy the traditional RED stability requirement, no matter how the value of $r_p$ changes, under no circumstances will the system become unstable. The Priority Checking mechanism does not bring instability to the system.

Next, we consider the stability region of PC-RED parameters.

From the simplified TCP/PC-RED block diagram, Figure 34, we can get the frequency response of the compensated loop transfer function as:
Because a large \( r_p \) can cause unacceptable congestion to flows without any QoS Requirement, in an ideal network, \( r_p \) should always less than half of the overall load level.

Since the average queue size generator in RED algorithm is acting as a low frequency filter in the system, the TCP/AQM system works in low frequency range. Thus we introduce a low frequency threshold \( \omega_g \) to degrade the impact of the two factors,

\[
\frac{j\omega}{2(1-r_p)N} + 1 \left( \frac{j\omega}{1 + \frac{1}{R_0}} \right)
\]

in the denominator. If we choose \( \omega_g = 0.1 \min \left\{ 2r_p N, \frac{1}{R_0^2 C}, R_0 \right\} \), we can have

\[
L(j\omega) \approx \frac{P_{\text{max}} R_0^3 C^3}{4r_p (1-r_p)(\text{max}_{\text{sh}} - \text{min}_{\text{sh}})N^2 \cdot e^{-j\omega R_0}}, \quad \forall \omega \in [0, \omega_g].
\]

Given any \( N \geq N^* \) and \( R \leq R^* \), the magnitude of the frequency response of the compensated loop transfer function should be:

\[
|L(j\omega)| \leq \frac{P_{\text{max}} (R^*)^3 C^3}{4r_p (1-r_p)(\text{max}_{\text{sh}} - \text{min}_{\text{sh}})(N^*)^2} \cdot \frac{\omega_g^2}{\sqrt{(\alpha C)^2 + 1}}
\]
Where $N^\prime$ is the lower bound of the network load level and $R^+$ is the upper bound of network delay. Based on Nyquist stability criterion [88], $|L(j\omega)|$ should be smaller than 1 and the argument of $L(j\omega)$ should be larger than $-180^\circ$.

As a result, the stability region of PC-RED parameters should be:

$$\frac{p_{\max}(R^\prime)^3 C^3}{4r_p(1-r_p)(\max_{\alpha}-\min_{\alpha})(N^\prime)^2} \leq \frac{\omega_s^2}{\sqrt{(\alpha C)^2 + 1}}$$

where

$$\omega_s = 0.1 \min \left\{ \frac{2r_p N}{R_0^2 C}, \frac{1}{R_0} \right\}$$

To analyze the relationship between $r_p$ and $N$ in a network with fixed link capacity $C$ and predefined RED parameters, we consider $r_p$ and $N$ as the only two variables.

If $r_p < \frac{RC}{2N}$, which gives us $\omega_s = 0.1 \frac{2r_p N}{R_0^2 C}$, from (4.5), we have the following stability requirement.

$$Ar_p - Ar_p^2 + N2r_p^3 - N2r_p^4 \geq B$$

$$A = 5R^3C^3\alpha$$

$$B = 5L_{\text{red}}^2\alpha(RC)^6C$$

Choosing $R = 0.2$ seconds, $C = 125$ packets/second, $p_{\max} = 0.1$, $\max_{\alpha} = 100$ packets and $\min_{\alpha} = 50$ packets. The weight factor for calculating average queue size is set to 0.0005. Since $N$ can not have a magnitude smaller than $10^0$, the condition (4.5) is always satisfied under our proposed network conditions.

If $r_p > \frac{RC}{2N}$, which gives us $\omega_s = 0.1 \frac{1}{R_0}$, from (4.5), we have the following stability requirement.

$$r_p - r_p^2 \geq \frac{25L_{\text{red}}^2\alpha^2(RC)^8}{4N^4(1+(10RC\alpha)^3)}$$
Choosing \( R = 0.2 \) seconds, \( C = 125 \) packets/second, \( p_{\text{max}} = 0.1 \), \( \text{max}_{\text{th}} = 100 \) packets and \( \text{min}_{\text{th}} = 50 \) packets. The weight factor for calculating average queue size is set to 0.0005. Since \( N \) cannot have a magnitude smaller than \( 10^0 \), and \( r_p \) only have an observable limitation when \( N \in [0, 0.028] \), we can still keep the system stable under all circumstances.

Theoretically, if the parameter setting follows (4.5), when the network condition and the RED parameters are defined as above, the system is stable regardless to the value of \( r_p \). When the network condition varies, the stability region of \( r_p \) may changes. However, this conclusion is based on the premise that the High-Priority flows only form a small portion of the entire work load, which makes \( \frac{2r_p N}{R_0 C} \) smaller than \( \frac{2(1-r_p)N}{R_0 C} \). Here we can give a range of \( r_p \) to be from 0 to 30%.

\[
r_{\text{pmax}} \in [0, 0.3]
\]  

(4.6)

Therefore, the QoSA-RED limits \( r_p \) by embedding an advanced packet scheduling mechanism. When the mechanism detects that the percentage of high-priority dataflows \( N_p \) has exceeded the ideal limit \( r_{\text{pmax}} \), it would redirect the packet to another path in the network. This algorithm will not be discussed further in the following content.

**B. w and System Performance**

Once the \( r_{\text{pmax}} \) is settled, by setting the parameters is accordance with (4.5), a stable system, related to both high-priority and non-priority dataflows, can therefore be constructed. However, the QoSA-RED is using the difference value between \( a_d \) and \( L \), the desired limit of average dropping rate \( a_d \), to determine the QoSA setting, and therefore to trigger the special dropping mechanism. The way that \( a_d \) fluctuates around \( L \) is also an important issue as the sensitivity of \( a_d \) directly impacts on QoSA-RED behaviour.

EWMA (Exponentially Weighted Moving Average) is used here as a low-pass filter to calculate the average-dropping rate \( a_d \). Thus the packet droppings caused by a short-
term increase of the average queue size will not result in a significant increase in the average-dropping rate.

If the $a_d$ is calculated in every standard dropping time interval $g$, and the packet size is fixed for the purposes of simplicity purpose, assume dropping occurs when the number of interval is $n$, and after $m$ intervals there is another dropping.

Thus the following equation can be developed:

$$
when \text{ there is no dropping action, } x_{(n)} = 0.
$$

$$
a_{d(n)} = (1 - w) \times a_{d(n-1)} + w \times x_{(n)} = (1 - w) \times a_{d(n-1)}
$$

$$
a_{d(n+1)} = (1 - w) \times a_{d(n)} + w \times x_{(n+1)} = (1 - w) \times a_{d(n)} = (1 - w)^2 \times a_{d(n-1)} \quad (4.7)
$$

when dropping occurs at time $t$, $x_{(n+m)} > 0$.

$$
a_{d(n+m)} = (1 - w)^m \times a_{d(n)} + w \times x_{(n+m)}
$$

Where

$$
m = \frac{t - t_n}{g}
$$

$t$ = the current packet dropping time;

$t_n$ = the last packet dropping time;

$g$ = the standard dropping time interval;

$w$ = the weight factor to calculate $a_d$;

$x_{(n+m)}$ = the total amount of dropped packets at time $t$.

Take one high-priority dataflow, for example, based on (4.4), and assume that the average-dropping rate is initially set to half of the desired limit $L$, the dropped packet is fixed as $X$ throughout the transmission and there is packet-dropping at every standard interval, thus we have:
\[ a_d(n) = f(w, n) \]

\[ = (1 - w)^n \times \frac{L}{2} + \sum_{i=0}^{n-1} w \times (1 - w)^i \times X \]

\[ = (1 - w)^n \times \frac{L}{2} + \frac{w \times (1 - (1 - w)^n)}{1 - (1 - w)} \times X \]

\[ = (1 - w)^n \times \frac{L}{2} + (1 - (1 - w)^n) \times X \]

\[ = (1 - w)^n \times \frac{L}{2} - (1 - w)^n \times X + X \]

Figure 44 shows the average-dropping rate \( a_d \) as the function of the weight factor \( w \) and the packet dropping counter \( n \), when the desired limit \( L \) is converted into size and set to be 300 and \( X \) is 500. It is shown that the ascending extent of \( a_d \) increases relative to the increase of the weight factor \( w \).

If \( w \) is too large, then the dataflow would be very sensitive to packet dropping. Thus, the probability of \( a_d \) exceeding the desired limit would also increase. This might be propitious to the transmission of high-priority dataflows. However, this might also lead to a complete loss of non-priority dataflows' transmission. A large \( N_p \) would result in large \( C_p(s) \) and \( C_n(s) \), which are the factors used to calculate the dropping probabilities \( p_p \) and \( p_n \). If \( w \) is too small, then \( a_d \) would not be a reasonable reflection of the current dropping action, as it would respond too slowly to the dropping. In this case, the QoSA-RED would not be able to detect the current level of QoS the high-priority dataflows are being treated with. In the scenario presented in Figure 40, if the QoSA-RED is expected to respond to the dropping action within 50 to 100 standard intervals, the \( w \) should be set in the range of approximately 0.003 to 0.03.
4.4 QoSA-RED NS Simulation

Our simulation is implemented in NS2. The aim is to analyse how the change in $w$ could affect the stability of the transmission of high priority dataflows and non-priority dataflows, and how the percentage of high priority dataflows $N_p$ in overall data load $N$, referred to as $r_p$, could influence the QoSA-RED QoS performance.

The experiment includes the following steps. First, we use the traditional RED to analyse the throughput of a network with a non-priority dataflow. Then we set some nodes with high priority and use the QoSA-RED to provide different levels of QoS. After that, we change the $w$ in the QoSA-RED to make the average dropping rates more or less sensitive to the dropping action, and study the system performance. Finally, we simulate the networks with 10%, and 30% high priority dataflows, respectively, to examine the effect of $r_p$.

We use the same single bottle-link network topology shown in Figure 24.
Simulation 2

Data sources consist of 20 FTP flows, and 60 HTTP sessions. Link capacity is 125 packets/s, which is 0.5 Mb/s with average packet size 500 bytes. The queuing buffer is 150 packets with minimum and maximum thresholds being 50 and 100 respectively. The average weight used to calculate the average queue size is set to be 1.33e-6. QoS Requirement carried in the header is a number $q_r$, which is converted into a percentage number and used as the maximum average dropping rate for the QoSA-RED to refer to while marking the QoS Alert.

Based on the stability requirement proposed in [21], we have:

$$L_{red} = \frac{p_{max}}{\max_h - \min_h} = 0.005$$

$$K = \frac{\log_e (1 - \alpha)}{\delta} = 1.66 \times 10^{-4}$$

$$\omega_g = 0.1 \min\left\{ \frac{2N^-}{(R')^2C}, \frac{1}{R'} \right\} = 1$$

Thus

$$\frac{L_{red}(R'C)^3}{(2N^-)^2} \leq \sqrt{\frac{w_g^2}{K^2}} + 1$$

is satisfied.

Based on the stability requirement in (4.5), with $r_p=0.3$, we have:

$$\omega_g = 0.1 \min\left\{ \frac{2r_pN}{R_0^2C}, \frac{1}{R_0} \right\} = 0.198347$$

$$\frac{p_{max}(R')^3C^3}{4r_p(1 - r_p)(\max_h - \min_h)(N^-)^2} = 0.12379$$

$$\sqrt{\frac{\omega_g^2}{(\alpha C)^2}} + 1 = 1193.066$$

So the stability requirement is also achieved.

In the last section, the relationship between the network condition and the suggested maximum High-Priority flows percentage has been worked out. Based on the suggestion,
the maximum number of High-Priority flows in this simulated network should be 31.25% of the overall work load to avoid instability in the queue length. The queue length plots of RED, QoSR-RED and QoSA-RED are shown in Figure 41. The dark line is the current queue size, while the light line is the average queue size.

Figure 41 Queue Length of RED, QoSR-RED and QoSA-RED in Simulation 2

Figure 42 shows the throughput of one FTP flow with QoS Requirement and one without under traditional RED, QoSR-RED and QoSA-RED. While there is no noticeable change in the throughputs of High-Priority flow under QoSR-RED and QoSA-RED during heavy congestion, the Non-Priority flow achieves a markedly larger throughput by using QoSA-RED.

By comprehensively considering the queue length and throughput, QoSA-RED is a better queue management method, which can provide higher level of QoS to high-priority dataflows than RED, while in the same time enhances the fairness among dataflows.

Simulation 3

In this simulation, we set $\alpha = 4e-3$ queue size 50 with minimum and maximum thresholds being 10 and 30 respectively, $w = 5e-3$ and $w = 3e-5$, respectively. We then compare the $a_\delta$ and throughput of the dataflow under these two conditions in Figure 43. For simplicity, we set the maximum average dropping rate of all High-Priority flow to 35%. The simulation duration is 3000 seconds. The light line is the desired dropping rate limit in both graphics on the left.
Figure 42 Throughput of High-Priority flow and Non-Priority flow in Simulation 2
When $w = 5e^{-3}$, it is sufficiently large to make the $a_d$ sensitive enough to detect the dropping action, which affects the data transfer. Therefore, by calculating the difference value between $a_d$ and $L$, the Priority List Maintenance Function can set the QoSA bit in time to inform the Pick Packet to Drop Mechanism. This action guarantees a certain level of quality of service to high-priority dataflows, and ensures a higher rank of throughput, which is around 8.5 KB/s as shown in Figure 43.
When $w$ is small as $(3e-5)$, the $a_d$ responds tardily to the dropping action. In this simulation, although the QoSA-RED drops packets from this high-priority dataflow throughout the entire 3000 seconds, it takes 2700 seconds for the $a_d$ to smoothly grow up to the desired limit $L$. And the QoSA-RED cannot help the high-priority dataflow to achieve a satisfying throughput over an extended period of time due to the lack of QoSA caution caused by the slow reaction of $a_d$. As a result, the throughput is less than one third of that produced with $w = 5e-3$.

**Simulation 4**

In this simulation we set $w = e-3$ and retained all of the other parameters identically to the previous experiment. The comparison of the throughput of the high-priority dataflow and the throughput of the non-priority dataflow are shown in Figure 48 and Figure 44, when $r_p$ is set to be 10% and 30%, respectively. The light line is the desired dropping rate limit in both graphics on the left.

When there is a 10% load level with a high-priority in the entire network, the throughput of high-priority dataflows is about 3.2 times the throughput of non-priority dataflows. The average dropping rate $a_d$ of high-priority dataflow is held around the desired limit 3.5%. However, the $a_d$ of the non-priority dataflows is 71% larger than that of the high-priority dataflow.
Figure 44 Average Dropping Rates and Throughputs of High-Priority and Non-Priority Flows when $r_p = 10\%$
When $r_p$ rises up to 30%, since the ratio of $N_p$ to $N$ is larger, the dropping probability of the high-priority dataflow would also increase. This certainly leads to a rise in the $a_d$ also. The system has reached its critical state due to the large $r_p$. Although the QoSARED priority checking function has been enabled, it is still unable to control the $a_d$ of the high-priority dataflow under the allowable dropping rate range $L$. The $a_d$ is 14% beyond the limit. And the $a_d$ of the non-priority dataflow is even higher and oscillates more acutely than previous, which leads to an extremely low level of throughput. Throughout
the simulation, the throughput of non-priority dataflow remains at an extremely low level, mainly zero, and goes up to 1KB/s occasionally. In addition, the throughput of high-priority dataflows is also lower than it was in a network with a smaller amount of high-priority dataflows.

4.5 Conclusion

The Priority Checking function is provided here as an AQM method for a network to support both high-priority dataflows and non-priority dataflows. The Priority Checking function does not involve the traditional RED parameters. But because of the new Priority List added in the RED, the extra function process times do affect the queue status and slightly increase the fluctuation when the performance is still stable. Therefore, PC-AQM is suitable for small scale network, as it only need to maintain a short Priority List.

The NS2 simulation shows that the PC-AQM has guaranteed a certain level of average dropping rate for high-priority dataflows, and helped the data source to achieve desirable transient performance. The influence of the average dropping rate weight factor and the percentage of high-priority dataflows in the overall load level have been studied.

It has been noticed that the QoSA-RED does provide a better throughput to FTP flows without QoS Requirement under heavy congestion. However, there is still a large probability that it may get worse when the network is dealing with a large portion of High-priority flows. How to guarantee the transient performance of non-priority dataflows in a network supporting PC-AQM is studied in the following chapter.
Chapter 5 Adaptative Priority Checking RED and Performance Analysis

Adaptive QoSA-RED (AQA-RED) and Gentle QoSA-RED (GQA-RED) are our next two PC-AQM algorithms. Their algorithms are studied and verified in this chapter. AQA-RED and GQA-RED both inherit the contributions of QoSA-RED, while AQA-RED gives further contribution on protecting Non-Priority dataflows under heavy congestion while guaranteeing certain level of QoS status to High-Priority dataflows, and GQA-RED protects certain level of QoS status to dataflows with high QoS Requirement in multi-congested-link networks.

5.1 Adapt QoSA-RED Parameters to network load level

In Chapter 4, two QoS related RED algorithms are introduced. QoSR-RED uses the IPv6 header to notify the router the QoS requirement of the end users. QoSA-RED, on the other hand, can not only inform the router the QoS requirement but also the current QoS status the dataflow has been handled in. Both QoSR-RED and QoSA-RED can guarantee the high-priority dataflow a better throughput than the non-priority dataflow under congestion. However, the non-priority dataflows can get over exacted under heavy congestion, and their throughputs can be seriously encroached on, because both QoSR-RED and QoSA-RED exhibit the essential characteristic of PC-AQM algorithms, which is undermining the QoS status of Non-Priority flows to guarantee higher QoS status for flows with QoS Requirements under congestion. In critical scenarios this can cause a QoS crisis for those end users running normal web applications to access basic contents and not willing to pay extra for it. This issue can be compensated for by introducing the adaptable maximum dropping probability or gently-varying QoS Alert to their working process.

In this chapter, two other QoSA-RED based priority related AQM algorithms are presented. The Adaptive QoSA-RED (AQA-RED) has a self-tuning function to tune the maximum dropping probability according to \( r_p \), the current ratio of high-priority dataflow to the overall workload. The Gentle QoSA-RED (GQA-RED) use three bits in the IP
header instead of one bit to mark the high-priority dataflow with different levels of QoS Alerts, and use these levels to generate their dropping probabilities during congestion. Both of these two algorithms can help to improve the QoS status of the non-priority dataflows under congestion.

5.1.1 Adaptive QoSA-RED (AQA-RED) Algorithms

Based on section 4.3.2, we understand that the $r_p$ should be controlled in a limited scale so as to keep system performance stable.

The main idea of implementing AQA-RED is to avoid the over restraint of the non-priority dataflow throughout throughout the transmission. The current dropping probability is determined not only by the maximum dropping probability and the average queue size, but also by the ratio of current $r_p$ to $r_{p\text{max}}$. The detailed algorithm is listed in Figure 46.

Upon every count-down timer interval, Maximum dropping probability is updated as follows:

If $(r_p \leq 0.25 \times r_{p\text{max}})$ then

If (average queue size < minimum threshold):

$p_{\text{max}} = p_{\text{max}} \times (1 - a \times r_p / r_{p\text{max}})$;

If (average queue size > maximum threshold):

$p_{\text{max}} = p_{\text{max}} \times (1 + a \times r_p / r_{p\text{max}})$;

Else If $(r_p \geq 0.75 \times r_{p\text{max}})$ then

If (average queue size < minimum threshold):

$p_{\text{max}} = p_{\text{max}} \times (b \times r_p / r_{p\text{max}})$;

If (average queue size > maximum threshold):

$p_{\text{max}} = p_{\text{max}} + c \times (r_p / r_{p\text{max}})$;

If ($p_{\text{max}} > 1$) then
$p_{\text{max}} = 1$;

Else If ($p_{\text{max}} < 0$) then
$p_{\text{max}} = 0$;

$p_{\text{max}}$: Maximum dropping probability used by Discard Tester;

$r_p$: The ratio of High-Priority flows to the overall work load;

$r_{p\text{max}}$: The suggested maximum $r_p$ of the network.

a: (0.095) $p_{\text{max}}$ updating parameter, used when $r_p$ is low.

b: (0.0935) $p_{\text{max}}$ updating parameter, used when $r_p$ is high.

c: (0.05) $p_{\text{max}}$ updating parameter, used when $r_p$ is high.

Figure 46 AQA-RED Maximum Dropping Probability Updating Algorithm
To reduce the computation, the calculation takes place every interval seconds rather than upon every packet arrival. When the system contains a small amount of High-Priority flows, the chance of getting over restraint of the Non-Priority flow throughput is also low. As a result, the Multiple Increase and Multiple Decrease (MIMD) method is used to tune the maximum dropping probability. When the average queue size is smaller than the minimum threshold, the maximum dropping probability will be decreased by the updating parameter \(a\) times \(\frac{r_p}{r_{p_{\text{max}}}}\) percent every interval seconds. But when the average queue has exceeded the maximum threshold, instead of dropping all of the incoming packets, we choose to increase the maximum dropping probability by the updating parameter \(a\) times \(\frac{r_p}{r_{p_{\text{max}}}}\) percent every interval seconds. But when the system contains a large amount of high-priority dataflows, we have to be more careful to increase the maximum dropping probability, as one small oscillation of \(p_{\text{max}}\) can lead to a large reduction of the throughput of the non-priority dataflows, as they are the minority in the system. The Additive Increase and Multiple Decrease (AIMD) method is used in this case. When the average queue size is under the minimum threshold, the maximum dropping probability is set to be the updating parameter \(b\) times \(\frac{r_p}{r_{p_{\text{max}}}}\) percents of its original value. When the average queue size exceeds the maximum threshold, the maximum dropping probability is increased by updating parameter \(c\) times the ratio of current \(r_p\) to \(r_{p_{\text{max}}}.\) The updating parameters here are constants.

**5.1.2 AQA-RED NS Simulation**

In this section, the QoSA-RED and AQA-RED behaviors are examined in two separate simulations. In Simulation 5 QoSA-RED is used to guarantee the QoS of one dataflow with High-Priority, while AQA-RED is used in Simulation 6 with all of the parameters and network settings remaining the same. We still use the single bottle link network topology of Figure 24 used in the previous simulations.

**Simulation 5**

The network topology used in Simulation 4 is reused here. We set the weight factor for calculating the average queue size to be \(5e-5\), and the weight factor for calculating the average dropping rate to be \(3e-3\). The suggested maximum \(r_p\) is set to be 30% according
to the stability requirement suggested in chapter 4. The duration of the simulation is set to
be 3000 seconds. There is only one High-Priority dataflow i performing as a one-way
long-live FTP flow with a high QoS Requirement throughout the simulation. FTP flow j
performs as a one-way long-live dataflow without any QoS Requirement. The other FTP
flows have varying QoS requirements so as to bring changes to the $r_p$. During $t = 0\text{--}750s$,
the percentage of the High-Priority flow is 0%, then rises from 0% to 10% at $t = 750s$,
and stays for another 750 seconds. At $t = 1500$ seconds, it goes up to 20% and at $t =
2250s$ it rises to 30% and stays for the rest of the duration. All of the QoS Requirements
of the High-Priority Flows are set to 50%.

The queue size is shown in Figure 47. The change of the average dropping rates and
the throughputs of dataflow $i$ and $j$ is shown in Figure 48 and Figure 49, respectively. The
dark line in Figure 48 indicates the average dropping rate, while the light line denotes the
desired average dropping rate limit.

![Queue size under QoSA-RED](image-url)
Chapter 5 Adaptive Priority Checking RED and Performance Analysis

Average Dropping Rate

![Average Dropping Rate Graph](image)

Figure 48 Average Dropping Rate of Flow i and Flow j

Throughput (KBytes/s)

![Throughput Graph](image)

Figure 49 Throughputs of Flow i and Flow j

Simulation 6

In this simulation, we use the AQA-RED instead of QoSA-RED to manage the queue. All of the other parameters remain the same. The queue size is shown in Figure 50.
Chapter 5 Adaptive Priority Checking RED and Performance Analysis

The change of the average dropping rates and the throughputs of dataflow \( i \) and \( j \) is shown in Figure 51 and Figure 52. The maximum dropping probability, which is updated every count down timer interval, is shown in Figure 53. The light lines in both graphics in Figure 51 denote the desired dropping rate limit.

![Queue Length (packets)](image)

**Figure 50** Queue size under AQA-RED

![Average Dropping Rate](image)

**Figure 51** Average Dropping Rate of Flow \( i \) and Flow \( j \)
Comparing the average queue sizes in both Figure 47 and Figure 50, we can tell that the oscillation has been significantly reduced with AQA-RED when there is 10% ~ 20% High-Priority flows in the network. However when the ratio increases to 30%, the queue performances of QoSA-RED and AQA-RED are alike. The high-priority dataflow $i$ has
achieved the desired throughput all the way through congestion, while the throughput of dataflow $j$ has also been better protected from being over encroached upon. However, when $r_p$ reaches a much higher level in the later part of the simulation, the oscillation becomes more obvious than it is in a network using QoSA-RED.

### 5.2 RED with Gentle QoS Alert Checking (GQA-RED)

To guarantee the transient performance of non-priority dataflows in a network supporting GQA-RED, we use a multi-bits QoSA instead of a one bit is used.

#### 5.2.1 Tagging algorithm at packet arrival

The tagging algorithm at each packet arrival is shown in Figure 54. If the QoSA field has not been updated by any other GQA-RED router yet, when the packet of a new dataflow arrives, the default value would be zero. The information carried in the QoSR field is a number in the range of 0 and 7, indicating the different ranks of average dropping rate the users are expecting. The QoSA also holds a value in the range of 0 and 7, which can be equally divided into four QoS Alert levels: LIGHT, MEDIUM, HIGH and CRITICAL. If the average dropping rate continues exceeding the QoS requirement upon every count-down timer interval, then the QoS Alert level would shift up from LIGHT to CRITICAL. Like-wise, the QoS Alert level could go down to LIGHT if a CRITICAL dataflow has an average dropping rate under the QoS requirement several time intervals in a row. In a DiffServ network, packets with the same QoS Alert level could be integrated and be handled in corresponding Per Hop Behaviours.

The end user can dynamically change the QoS Requirements depending on the amount of money he or she is willing to pay for the network services. The GQA-RED will notice this change once the packet with the new QoSR arrives.

Because the QoSA in the Priority List is actually a sum value of the $q_a$ that the current router is holding and the value that the packet is carrying, the GQA-RED router can track the QoS status of one dataflow throughout the transactions in the network.
Chapter 5 Adaptive Priority Checking RED and Performance Analysis

Upon each packet arrival, Priority list updates its state variables as follows:

1. Check whether it carries a new Flow ID;
   - If Flow ID is new, then
     - Insert a new entry to the list; \( f_{id} = \text{Flow ID} \);
     - Get the QoS status from the QoSA field; \( q_{a} = \text{QoSA} \);
     - Get the QoS requirement from the QoSR field; \( q_{r} = \text{QoSR} \);
     - \( a_{d} = 0; \ n_{d} = 0; \ n_{a} = \text{new packet size} \);
   - Else
     - \( n_{a} = n_{a} + \text{new packet size} \) (where \( f_{id} = \text{Flow ID} \));
     - \( q_{r} = \text{QoSR} \);
     - \( q_{a} = q_{a} + \text{QoSA} \);
     - Tag the QoSA field with the updated \( q_{a} \).

Upon every count-down timer interval, Priority list updates its state variables as follows:

- \( a_{d} = a_{d} \ast (1 - w) + w \ast (n_{d} / n_{a}) \);
- \( n_{d} = 0; \ n_{a} = 0 \);
- Compare Average Dropping Rate with the labeled QoS Requirement;
  - If \( (a_{d} \geq q_{r}) \) then
    - \( q_{a} = +; \) (Maximum value: 7)
  - Else
    - \( q_{a} = -; \) (Minimum value: 0)

Figure 54 GQA-RED Tagging Algorithm

5.2.2 Buffer management and packet dropping algorithm in GQA-RED

After tagging the incoming packet, the GQA-RED router will then monitor buffer occupancy and determine whether this received packet should be dropped or not. The pseudo-code is shown in Figure 55.

When the average queue size is between the two thresholds, the dropping probability of the received packet is inversely proportional to its QoSA value. Thus, a packet carrying a LIGHT QoSA value would more likely to be dropped compared to a packet with a CRITICAL QoSA field.

This mechanism allows the router to use only one set of RED parameters to calculate the overall dropping probability \( P_{\text{red}} \) from the average queue length. However, when breaking the incoming dataflows into different categories based on the tagged QoSA value, \( P_{\text{red}} \) can be divided into differentiated dropping probabilities. Thus GQA-
Chapter 5 Adaptive Priority Checking RED and Performance Analysis

RED can provide different levels of QoS to dataflows with different QoSA by manipulating their dropping probabilities respectively.

Upon receiving one packet:
- If (average queue size > Max threshold) then
  Drop the packet;
  **Else if** (average queue size > Min threshold) then
  Calculate $p_{red}$, the dropping probability as traditional RED does;
  $$P_{QoSA-RED} = \frac{3}{1 + qa};$$
  If $p_{QoSA-RED}(kT) < 0$ then $p_{QoSA-RED}(kT) = 0$;
  If $p_{QoSA-RED}(kT) > 1$ then $p_{QoSA-RED}(kT) = 1$;
  Drop the packet with probability $p_{QoSA-RED}$;
  **Else**
  Accept the packet;
  - $T$ is the sampling time.
  - $k$ is the sample instant.

Upon dropping one packet, Priority list updates its state variables as follows:
- $n_d - n_d +$ dropped packet size (where $f_d = Flow ID$);

Figure 55 GQA-RED Buffer Management and Packet Dropping Algorithm

5.2.3 GQA NS Simulation

The aim is to examine not only the GQA-RED’s ability to respond to end user’s QoS requirements, but also its ability to record the QoS status throughout the transmission in complex networks. The network topology that we use is shown in Figure 60. It is a multi-congestion-link network. Source i sends dataflows with high QoS Requirements, while source j sends dataflows with none. All the way through the route from source i, j and the destination node i' and j', there are other sources and destinations sharing every single link.
Chapter 5 Adaptive Priority Checking RED and Performance Analysis

Simulation 7

We set the number of core routers to be 4. All links between core routers are congested. Dataflows $i$ and $j$ traverse more than one congested link and other flows, such as Flow $2_n$ and Flow $3_n$, traverse only one congested link. Each propagation delay is set to be 100ms. Every link between core routers is shared by $i$, $j$ and 8 other Non-Priority FTP flows along with 60 HTTP sessions. The simulation lasts for 3000 seconds. QoSA-RED is used in all of the core routers. The queue length and throughput of Flow $i$ and $j$ are shown in Figure 57 and Figure 58, respectively. The light line in Figure 57 is the average queue size, and the dark line is the current queue size.
Chapter 5 Adaptive Priority Checking RED and Performance Analysis

As the only flow with QoS Requirement in the network, Flow i receives a much higher throughput without overly undermining the QoS of the regular dataflows.

**Simulation 8**

We now use GQA-RED instead of QoSA-RED in all of the core routers. Figure 59 shows the current queue size and average queue size under GQA-RED. The throughputs of Flows i and j are shown in Figure 60. The average dropping rate under both QoSA-
RED and GQA-RED are shown in Figure 61. The QoS Alert of High-Priority dataflow is also shown in Figure 61.

![Queue Length under GQA-RED](image1)

**Figure 59** Queue Length under GQA-RED

![Throughputs of Flow i and Flow j under GQA-RED](image2)

**Figure 60** Throughputs of Flow i and Flow j under GQA-RED
Chapter 5 Adaptive Priority Checking RED and Performance Analysis

![Figure 61: Average Dropping Rates and QoS Alert under GQA-RED](image)

Compared to the throughput of the High-Priority Flow in Simulation 7, the throughput of Flow i slightly rises while the throughputs of flows without QoS Requirements remain at the same level. Unlike AQA-RED, the demands placed on extra computation in GQA-RED is not that great. When implemented in complex network topologies, GQA-RED also has the ability to track the overall QoS status of one flow, which is also one of the reasons it guarantees better throughput to the High-Priority Flows.

When the intensity of congestion in the network changes, the difference between the performance of QoSA-RED and GQA-RED varies as well.

5.3 Conclusion

Our algorithm for Adaptive QoSA-RED seems to work well accomplishing the mission of guaranteeing QoS to special dataflows as well as providing acceptable throughput to dataflows without a QoS Requirement. As a result of the slow adaptation of the maximum dropping probability, the design of Adaptive QoSA-RED gives robust performance in a wide range of environments. As stated above, the cost of this slow adaptation is that of a transient period, after a sharp change in the level of congestion, when the average queue size is not within the target zone. Adaptive QoSA-RED is thus consciously positioned in the conservative, robust end of the spectrum of AQM.
mechanisms, with the aim of avoiding the more finely-tuned but also more fragile dynamics at the more aggressive end of the spectrum.

The Gentle QoSA-RED algorithm guarantees the transient performance of non-priority dataflows by tracking the QoS status throughout the route and grading it into 8 different levels. The simulations show that the non-priority dataflows have been better protected compared to in a QoSA-RED network.

Both AQA-RED and GQA-RED have achieved the goal of serving dataflows with certain levels of QoS based on the end users demands. However, the slower response and slightly larger oscillation caused by the priority checking function and dropping probability calculation function, as well as the parameter tuning problem inherited from traditional RED are their common shortcomings. In the next chapter, the priority checking function is used in conjunction with the traditional PI controller for AQM to improve the system performance.
Chapter 6  Proportional Integral Controller with Priority Checking Function

In this chapter, we have combined PI-AQM and GQA-RED to develop two new PC-AQM algorithms, Priority Checking PI controller (PCPI) and Double Status PCPI, both of which not only inherit the contributions of GQA-RED, but have much smaller response time. In addition, compared to PCPI, DSPCPI has the ability to remove the overshoot, hence stabilize the QoS status when network load level varies.

6.1 Proportional Integral Controller for Queue Management

In industrial control systems, a Proportional–Integral–Derivative controller (PID controller) \[89\] is a generic control loop feedback mechanism which is widely used. It attempts to correct the error between a measured process variable and a desired set-point by calculating and then outputting a corrective action that can adjust the process accordingly.

The PID controller algorithm involves three separate parameters: Proportional factor, Integral factor and Derivative factor. The Proportional factor \(K_p\) determines the reaction to the current error, the Integral factor \(K_i\) determines the reaction based on the sum of recent errors, and the Derivative factor \(K_d\) determines the reaction to the rate at which the error has been changing. They will then be weighted based on the gravity of their impact upon the system and used to adjust the process via a control element, such as the position of a control valve on an outgoing water pipe, or, in the case of queue management, the dropping probability of the incoming packet.

Some control systems may require only one or two modes to provide an appropriate level of system control. This is achieved by setting the gain of undesired control outputs to zero so as to ignore one or two components in the controller. A PID controller can transform into a PI, PD, P or I controller in the absence of the respective control actions. The derivative action is very sensitive to measurement noise, and the absence of an
integral value may prevent the system from reaching its target value due to the control action. As a result, PI controllers are particularly common.

In the previous two chapters, four algorithms related to Priority Checking are introduced. All of them are based on the traditional RED queue management algorithm. Although RED is very widely accepted, it still has some limitations. The first limitation deals with the trade-off between speed of response and stability. A design that is fast in its response time, is found to have relatively low stability margins, while a design that is stable exhibits sluggish responses. The other limitation of RED is the direct coupling between queue length and loss probability. The steady state queue length in RED is dependent on the load level. Hence, for an overloaded system, the flows pay a double penalty of higher delay as well as higher loss. The two can be easily decoupled.

Based on the linearized model, the design rules for obtaining a stable linear feedback control system with RED controller is shown in chapter 4. A limitation of RED design (inherent in the nature of the RED controller) is that the response time of the control system is quite long. Specifically, the response time of the system is limited to $1/\omega_s$ sec, where

$$\omega_s = 0.1 \min\{\rho_{pop}, p_{queue}\}$$

The multiplication factor of 0.1 is the trade-off between stability margins and speed of response. Larger values than 0.1 yield more responsive designs. However they have lower stability margins.

Intuitively speaking, the lag introduced by the low pass filter is a cause of the sluggishness of the response. One way to improve the response time of the system is to remove the low pass filter, and introduce what is known as the classical proportional integral controller.

A brief introduction of the Proportional Integral Controller for queue management is given in section 3.6. The PI AQM controller has a transfer function of the form...
Chapter 6 Proportional Integral Controller with Priority Checking Function

\[ C(s) = K_{pi} \frac{(\frac{s}{C} + 1)}{s} \quad (6.1) \]

A desired consequence of the Integral factor in \( C(s) \) is that \( \delta q \) in Figure 19 can asymptotically converge to zero if \( C(s) \) stabilizes \( P(s) \). When considering it as a TCP/AQM plant controlled by the PI controller, we need to choose the location of the zero \( z \) and the value of the PI factor \( K_{pi} \). Based on the proposition presented in [37], we consider the setup studied in Example 1 in the classic paper for analyzing RED [21], where \( C = 3750 \) packets/sec, \( N = 60 \), \( q_0 = 175 \) and \( R = 0.246 \) sec.

As
\[ \frac{2N^-}{(R^+)^2 C} = 0.5288 \quad \text{and} \quad \frac{1}{R^+} = 4.065 \]
the equation:
\[ \frac{2N^-}{(R^+)^2 C} \ll \frac{1}{R^+} \]
Based on the stability requirement of designing PI controller suggested in [37], with
\[ \omega_g = \frac{2N^-}{(R^+)^2 C} = 0.5288 \]
let
\[ K_{pi} = \omega_g \left[ \frac{(\frac{j\omega_g}{P_{queue}} + 1)}{(R^+)^3 C} \right] = 9.6426e - 6 \]
Thus,
\[ C_{pi}(s) = 9.6426e - 6 \frac{(\frac{s}{C} + 1)}{0.5288} \]
This PI compensator can stabilize the feedback control system for all \( N \geq N^* \) and all \( R_0 \leq R^* \).

According to [37], for such system with \( \omega_g = 0.5 \) rad/sec, the queue length should be sampled at approximately 10 to 20 times \( (\omega_g)/2\pi \), which is about 3-6 Hz. However, in RED algorithm, since the computation happens upon every packet arrival, when the link
capacity is 3750 packet/sec, the computation has to be carried out at 3750 Hz. Thus PI controller can speed up the computation by around 3 orders of magnitude.

6.2 Priority Checking PI Controller (PC-PI)

To dataflows with different priorities, the Proportional Integral controller can work with Priority Checking function to divide the dropping probability into different ranks.

The tagging algorithm of PC-PI is rather similar to PC-AQM algorithms introduced in the previous chapters, apart from the buffer management and packet dropping function. As one of the AQM algorithms with Priority Checking function, PC-PI also has a priority list to maintain upon every count-down timer interval.

| Upon each packet arrival, Priority list updates its state variables as follows: |
| Check whether it carries a new Flow ID; |
| If Flow ID is new, then |
| Insert a new entry to the list; \( f_{id} = \text{Flow ID}; \) |
| Get the QoS status from the QoSA field; \( q_a = \text{QoSA}; \) |
| Get the QoS requirement from the QoSR field; \( q_r = \text{QoSR}; \) |
| \( a_d = 0; n_d = 0; n_a = \text{new packet size}; \) |
| Else |
| \( n_a = n_a + \text{new packet size (where } f_{id} = \text{Flow ID}); \) |
| \( q_r = \text{QoSR}; \) |
| \( q_a = q_a + \text{QoSA}; \) |
| Tag the QoSA field with the updated \( q_a \); |
| Upon every count-down timer interval, Priority list updates its state variables as follows: |
| \( a_d = a_d * (1 - \omega) + \omega * (n_d / n_a); \) |
| \( n_d = 0; n_a = 0; \) |
| Compare Average Dropping Rate with the labeled QoS Requirement; |
| If \( a_d \geq q_r \) then |
| \( q_a ++; \) (Maximum value: 7) |
| Else |
| \( q_a --; \) (Minimum value: 0) |

Figure 62 PC-PI Tagging Algorithm
Upon receiving one packet:

\[ p_{pi}(kT) = a_{pi} \cdot (q(kT) - q_{ref}) - b_{pi} \cdot (q((k-1)T) - q_{ref}) + p_{pi}((k-1)T) \]

\[ p_{pc-pi}(kT) = p_{pi}(kT) \cdot \left( \frac{3}{1 + q_{a}} \right); \]

If \( p_{pc-pi}(kT) < 0 \) then \( p_{pc-pi}(kT) = 0; \)

If \( p_{pc-pi}(kT) > 1 \) then \( p_{pc-pi}(kT) = 1; \)

Else

Drop the packet with probability \( P_{pc-pi}; \)

Upon dropping one packet, Priority list updates its state variables as follows:

\[ n_{a} = n_{a} + \text{dropped packet size (where} f_{a} = \text{Flow ID);} \]

Figure 63 PC-PI Buffer Management and Packet Dropping Function

The difference between PC-PI and GQA-RED is that the former does not have two queue size threshold \( \min_{th} \) and \( \max_{th}. \) The \( q_{ref} \) is considered as the set-point in the traditional PI controller, which is the target queue size. The PC-PI controller first generates a generic dropping probability \( P_{pi} \) for all incoming packet based on the difference between the queue size and the \( q_{ref}. \) \( P_{pi} \) is then used in conjunction with the QoS Alert carried in the IPv6 header to generate the dropping probability \( P_{pc-pi} \) to each chosen packet.

6.3 PC-PI NS Simulation

The equation (6.1) is the transfer function of PI controller in s-domain. To implement it in NS2, we first use a z-domain transfer function first.
Chapter 6 Proportional Integral Controller with Priority Checking Function

\[ C_{PI}(s) = K_{PI} \left( \frac{s}{\omega_g} + 1 \right) = \frac{K_{PI}}{\omega_g} + \frac{K_{PI}}{s} \]

Set

\[ s = \frac{2}{T} \frac{1-z^{-1}}{1+z^{-1}} = \frac{2}{T} \frac{z-1}{z+1} \]

\[ C_{PI}(z) = \frac{K_{PI}}{\omega_g} + \frac{K_{PI}}{2} \frac{z-1}{z+1} = \frac{\left( \frac{K_{PI}}{\omega_g} \frac{TK_{PI}}{2} \right) z - \left( \frac{K_{PI}}{\omega_g} \frac{TK_{PI}}{2} \right)}{z-1} = \frac{a_p z - b_p}{z-1} = \frac{p(z)}{\delta q(z)} \]

where

\[ K_{PI} = \omega_g \sqrt{\frac{\left( j \omega_g + 1 \right)}{p_{queue}}} \]

\[ \omega_g = \sqrt{\frac{2N^+}{(R^+)^2 C}} \]

\[ T = \frac{1}{\text{sampling frequency}} \]

Thus, the difference equation of the variables yielding, at time \( t = kT \),

\[ p(kT) = \left( \frac{K_{PI}}{\omega_g} + \frac{TK_{PI}}{2} \right) \delta q(kT) - \left( \frac{K_{PI}}{\omega_g} - \frac{TK_{PI}}{2} \right) \delta q((k-1)T) + p((k-1)T) \]

Considering the single bottle link network topology of Figure 24 used to verify QoSA-RED algorithms in the previous chapter, we give new parameters to it:
Chapter 6 Proportional Integral Controller with Priority Checking Function

\[ N^- = 60, R^+ = 0.25s, C = 2500 \text{ packet/s}, T = 1/200Hz = 5ms \]

\[ \omega_s = \frac{2N^-}{(R^+)^2C} = 0.768 \text{ rad/sec} \]

\[ K_{pi} = \omega_s \left( \frac{j\omega_s + 1}{P_{queue} + \left( R^+ C \right)^3} \right) = 4.613e-6 \]

\[ a_{pi} = \frac{K_{pi}}{\omega_s} + \frac{TK_{pi}}{2} = 6.016e-5 \]

\[ b_{pi} = \frac{K_{pi}}{\omega_s} - \frac{TK_{pi}}{2} = 5.994e-5 \]

Since this setting also satisfies the stability equation \[ \frac{2N^-}{(R^+)^2C} \ll \frac{1}{R} \], we use it for any N larger than N^- and any R smaller than R^+ as the stable controller to the system. The queuing performance of PI and PC-PI controller is first presented in Simulation 9. To validate the performance of the PI controller, we implemented it in NS with a sampling frequency of 200 Hz. Thus the PI coefficients band implemented, \( a \) and \( b \), were 6.016e-5 and 5.994e-5 respectively.

**Simulation 9**

The PI controller gives much better performance than RED-Based algorithms. This fact has been proved in many simulations done by other researchers. Since the minimum number of flow is \( N^- = 60 \) and the maximum value of propagation delay is \( R^+ = 250 \text{ms} \), we set the network with the time varying dynamics and the mixture of 180 FTP and 100 HTTP flows. The propagation delays are uniformly around 50ms, with the average packet size being 500 Bytes. The QoSA-RED controller in Simulation 1, PI controller and PC-PI controller are also used. During \( t = 0 \sim 1000s \), 180 FTP flows are active, At \( t = 1000s \), one third of the flows drop out of the network and re-join the network at \( t = 2000s \). For QoSA-RED we set the minimum and maximum thresholds to be 80 and 150 respectively.

103
Maximum dropping probability is 0.1. $q_{ref}$ for the PI controller was chosen to be 100 packets. Queue Buffer is 400 packets.

The total duration of this simulation is 3000 seconds. The queue length plots for the three controllers are depicted in Figure 64, where the faster response time as well as the regulation of the output to a constant value by both PI and PC-PI controls are clearly observed. Unlike QoSA-RED, both of PI and PC-PI controllers are largely insensitive to the load level variations and attempt to regulate the queue length to the same value of 100. However, due to the large sized Priority List and the increment of the computation required to maintain it, the oscillation of the PC-PI is larger than under PI controller.

![Queue lengths](image1)

**Figure 64** Queue Sizes under QoSA-RED PI and PC-PI in Simulation 9

![Current dropping probabilities](image2)

**Figure 65** Current Dropping Probabilities under QoSA-RED PI and PC-PI in Simulation 9
Figure 66 Throughputs under QoSA-RED PI and PC-PI in Simulation 9

Figure 65 and Figure 66 shows the average dropping rate of dataflows and the throughputs of them under QoSA-RED and PC-PI. It is observed that under PC-PI the High-Priority dataflow experiences a throughput with smaller fluctuation than under QoSA-RED.

Simulation 10

Now we increase the number of FTP flows to 360 and HTTP flows to 200. QoSA-RED and PC-PI controller are compared. By our analysis, the performance of the controllers should slow down for higher load levels (gain N). The queue sizes are plotted in Figure 71 and we observe significantly better performance from the PC-PI controller. The QoSA-RED controller takes even a longer time to settle down, with the equilibrium queue length quite large compared to the last experiment. PC-PI also experiences a large
queue size at the beginning of the simulation. This is a result of the fact that at such high load levels, the loss probability has become so high that the steady state regulation error of those two controllers has pushed the operating queue length beyond the buffer size. However both of the controllers are still guaranteeing the throughput of High-Priority dataflow a larger value as can be seen in Figure 68. The PC-PI controller appears to be more robust in the face of higher loads. Considering throughput presented in Figure 72, PC-PI controller seems to be a better choice than QoSA-RED to dataflows with special QoS Requirements.

![Queue Sizes under QoSA-RED and PC-PI in Simulation 10](image)

Figure 67 Queue Sizes under QoSA-RED and PC-PI in Simulation 10
Chapter 6 Proportional Integral Controller with Priority Checking Function

Simulation 11

We have stretched the controllers a lot in the previous simulation. It is mentioned that the minimum number of flow should be \( N^- = 60 \) and the maximum value of propagation delay should be \( R^+ = 250 \text{ms} \) for the PI controller to satisfy all of \( N > N^- \) and \( R < R^+ \). In this simulation we reduce \( N \) into 20 FTP flows, and use all of the parameters presented in Simulation 2 to build our network. Data sources consist of 20 FTP flows, and 60 HTTP sessions. Link capacity is 125 packets/s, which is 0.5 Mb/s with average packet size 500 bytes. The queuing buffer is 150 packets with minimum and maximum thresholds being 50 and 100 respectively. The average weight used to calculate the
average queue size is set to be 1.33e-6. The two plots are shown in Figure 69. As we can observe, the PC-PI controller continues to exhibit acceptable performance, although it has become a little slower in its response time. The QoSA-RED maintains the queue size within the thresholds as well. The PC-PI controller does achieve a better result than QoSA-RED while guaranteeing the High-Priority flow throughput as can be seen in Figure 70. However, the oscillation is much larger than the one in Simulation 10. From the throughput point of view, the PC-PI still accomplished the mission of guaranteeing the QoS Requirement better than QoSA-RED.

![Figure 69: Queue Sizes under QoSA-RED and PC-PI in Simulation 11](image)

Figure 69 Queue Sizes under QoSA-RED and PC-PI in Simulation 11
6.4 DSPCPI Controller

PC-PI works satisfyingly when the network load level is stable. However, when there are many dataflows dropping out of the network or transmitted into the network, large oscillations of the queue size occur. Hence, the overall throughput would reduce, which leads to a larger number of dataflows carrying QoS Alert warning. To improve the QoS under unstable work load, we propose Double Status Priority Checking PI controller (DSPCPI).
The algorithm of DSPCPI is rather simple. PI controller is used when the work load is stable, so as to manage the queue size in a certain range. When there is large fluctuation of the work load, we use a different controller to rapidly reduce the system response time. The detailed algorithm is presented as follow.

Upon each count down timer interval:
- \[ q_{\text{ave}} = (1 - \alpha) \times q_{\text{ave}} + q_{\text{cur}} \times \alpha \]
- if \[(1+0.25) \times q_{\text{ref}} < \text{queue buffer}\]
  - \[ q_{\text{sw}} = q_{\text{ref}} + 0.25 \times q_{\text{ref}} \]
else
  - \[ q_{\text{sw}} = q_{\text{ref}} + \frac{\text{queue buffer} - q_{\text{ref}}}{2} \]
- \[ q_{\text{ref}} = q_{\text{ref}} - 0.08 \times q_{\text{ref}} \]

If average queue size is under the switch queue size \( q_{\text{sw}} \): Consider the status as Normal; Calculate the dropping probability as PC-PI does;
Else
  - Consider the status to be Critical;
- \[ k_1 = \frac{k_2 \times (q_{\text{cur}} - q_{\text{ref}2})}{1 + p_{\text{cri}}} \]
- \[ p_{\text{cri}} = k_1 \times (q_{\text{ave}} - q_{\text{ref}2}) \]

- \( q_{\text{ave}} \): average queue size, updated every packet arrival;
- \( q_{\text{cur}} \): current queue size;
- \( \alpha \): the queue weight, usually set as 0.002;
- \( q_{\text{sw}} \): the switch between two control statuses;
- \( q_{\text{ref}} \): the target queue size under Normal control status;
- \( q_{\text{ref}2} \): the target queue size under Critical control status;
- \( k_1 \): dropping probability updating variable;
- \( k_2 \): dropping probability updating parameter, \( 5e - 4 \).
- \( p_{\text{pi}} \): dropping probability used under Normal control status;
- \( p_{\text{cri}} \): dropping probability used under Critical control status;

Figure 71 DSPCPI Dropping Probability Updating Algorithm

The main idea of using two control statuses is to fasten the response time when the queue length is out of reasonable range due to the rapid change of the work load in the network. When there is a large amount of flows leave the network, since it will not cause
any congestion, the DSPCPI does not response to that. Hence the switch between two control statuses can only be triggered when the average queue size exceed it. We use average queue size instead of current queue size is because while the work load of the network changes, the queue size fluctuates with it. While the current queue size is rather unstable, bringing a low-pass filter into the system can reduce the oscillation, hence gives the system enough time to response to the new dropping probability.

\( k_1 \) is the dropping probability updating variable. It is proportional to the difference value between the current queue size and the target queue size. Since we do not want DSPCPI to over-drop packets, when taking the previous dropping probability into account, we suggest \( k_1 \) should be reduced by a certain rate related to \( p_{ori} \).

6.5 DSPCPI NS Simulation

We now compare the performances of DSPCPI with PC-PI under the same scenarios presented in Simulations 9 10 and 11.

Simulation 12

The setting of this simulation completely reused the setting in Simulation 11. We use DSPCPI controller instead of QoSA-RED and PI controller to compare the performance with PC-PI controller. The queue length plots for the PC-PI and DSPCPI controllers are depicted in Figure 72. It is observed that both of the controllers have the response time faster that QoSA-RED response time in Simulation 9. DSPCPI controller responses even faster than PC-PI controller, and does not have an overshoot at all. Both of them regulate the queue length to the value of 50 with an oscillation of ± 25 packets. However, compared to the latter, the former is much more insensitive to the load level variations when one third of the flows drop of and rejoin the network. The throughputs under PCPI controller are shown in Simulation 11. Figure 73 shows the throughputs under DSPCPI.
Chapter 6 Proportional Integral Controller with Priority Checking Function

Figure 72 Queue Sizes under PC-PI and DSPCPI in Simulation 12

![Queue Length (packets) Comparison](image)

Figure 73 Throughputs under DSPCPI in Simulation 12

![Throughput Comparison](image)

Simulation 13

Now we let half of the dataflow drop off the network at $t = 1000$ seconds and rejoin at $t = 2000$ seconds to cause some oscillation of the load level. The queue sizes are plotted in Figure 78 and we observe significantly better performance from the DSPCPI controller.
Figure 74 shows the queue size under PC-PI and DSPCPI controllers in Simulation 13. Figure 75 shows the current dropping probability under PC-PI and DSPCPI controllers. When the load level reduces to half of its original value, DSPCPI does not perform differently than PCPI. This is because only when the queue size exceeds the desired limit, could we consider there is congestion in the link, thus trigger the second status of the controller in DSPCPI. It is not necessary invoke extra computation when the probability of the queue getting jammed is rather small.
Simulation 14

In this simulation, we analyze the influence caused by different time delays to PC-PI and DSPCPI. The gains of Proportional Integral Controller are set to the same value as in simulation 12. The network consists of 20 FTP flows. Link capacity is set to be 125 packets/second. The delays are set to be 0.5 seconds, 0.8 seconds and 1 second, respectively. The plots of queue sizes are shown in Figure 77.
As we can see, both of the controllers achieve very unstable queue size throughout the simulation. Although DSPCPI does have a slightly smaller queue size, it is still hard to describe it as a stable system output.

6.6 Conclusion

Two alternative Priority Checking AQM controllers to QOSA-RED, PC-PI and DSPCPI, are presented here. The former provides improved network performance. The latter solves the problem of slow response in PC-PI under work load burst. Both controllers respond faster than the RED-Based controller with Priority Checking mechanism while DSPCPI was superior in robustly regulating the steady-state value of the queue level.
For Priority Checking AQM performance, we focused on objectives including queue usage and differentiated QoS. These were achieved in the simulations. Our approach to PC-AQM control was deliberately simple and straightforward. However, compare to traditional PI controller, the PC-PI and the DSPCPI are both slightly inferior on the speed and stability due to the extra computation. In the next chapter, we will introduce the prediction mechanism to these algorithms so as to make up the system response.
Chapter 7 Active Queue Management Algorithms with Prediction Mechanism

This chapter presents our last PC-AQM algorithm, KPI, which is developed from PC-PI controller and uses a Kalman predictor to predict the queue size. KPI inherits the contributions from the previous PC-AQM algorithms. In addition, it compensates for the time delays caused by extra computation required by those extra components in PC-AQM algorithms.

7.1 Kalman Predictor

Based on the mathematical models of TCP/AQM networks, several conventional controllers, such as PI, sliding mode [90] and Coefficient Diagram Method [91] have been designed as AQM methods in TCP networks. When adding the extra Priority Checking and Packet Header Labeling mechanism into the router, the delay of the network increases. However, most of the controllers mentioned above do not deal with the stochastic behavior and the inherent delay of the computer networks.

The work [92] gives a complete stability region characterization according to network load level, link capacity and network delays. However, choosing different coordinates within the same region can lead to different system performance as well. If we could bring prediction into the algorithm, the influence of network delays to the system could be minimized.

In control theory, Model predictive controllers [93] have been categorized as advanced control strategy and used widely in many industries. But it is computational expensive and applied usually on slow dynamic systems. The bottleneck of implementing prediction mechanism in the router is probably the computational complexity.

Most filters used in control practice, such as a low-pass filter, are formulated in the frequency domain and then transformed back to the time domain for implementation. The Kalman filter [94] is a time-domain efficient recursive filter that estimates the state of a
dynamic system from a series of noisy measurements. In contrast to batch estimation
techniques, no history of observations or estimates is required for Kalman Predictor. Only
the estimated state from the previous time step and the current measurement are needed
to compute the estimate for the current state. The Kalman filter model assumes the true
state at time $k+1$ $X_{k+1}$ is evolved from the state at $k$ according to:

$$X_{k+1} = AX_k + Bu_k + w_k$$
$$Y_k = CX_k + v_k$$

where

$X_k \in \mathbb{R}^n$, $u_k \in \mathbb{R}^m$, $w_k \in \mathbb{R}^q$, $Y_k \in \mathbb{R}^p$, $v_k \in \mathbb{R}^p$

$u_k$ deterministic, $X_0$ Gaussian, $X_0 \sim N(E[X_0], P_0)$

{w_k} and {v_k} are white Gaussian noises, mutually independent,
and independent of $X_0$, $w_k \sim N(0, Q_k)$, $v_k \sim N(0, R_k)$

$Y_k$ is an observation or measurement of the true state $X_k$ at time $k$. $u_k$ is the control
input. In many applications it is zero. $w_k$ is the process noise, which is assumed to be
drawn from a zero mean multivariate normal distribution with covariance $Q_k$. $v_k$ is
measurement noise, which is assumed to be drawn from a zero mean multivariate normal
distribution with covariance $R_k$. For simplicity of notation, $A$, $B$ and $C$ are time-invariant.
A is the state transition model which is applied to the previous state. B is the control-input model which is applied to the control vector $u_k$. C is the observation model which
maps the true state space into the observed space. Gaussian assumption is not required.
But it makes the Kalman estimator an optimal estimator. Many real dynamical systems
do not exactly fit the model. However, since Kalman filter is designed to operate in the
presence of noise, an approximate fit is often good enough for the filter to be very useful.
Variations on the Kalman filter may allow richer and more sophisticated models.

Kalman filters work in two distinct phases. It first predicts an estimate of the status
at the current time step using the state estimate from the previous time step, then update
the state estimate of the current time step through refining this prediction based on the
measurement information at the current time step.

The prediction equations are:
Active Queue Management Algorithms with Prediction Mechanism

\[
\hat{X}_{j|k-1} = A_k \hat{X}_{j|k-1} + B_k u_k \quad \text{(Predicted state)}
\]
\[
P_{j|k-1} = A_k P_{j-1|k-1} A_k^T + Q_k \quad \text{(Predicted Estimate Covariance)}
\]

The update equations are:

\[
\tilde{z}_k = Y_k - C_k \hat{X}_{j|k-1} \quad \text{(Innovation or measurement residual)}
\]
\[
S_k = C_k P_{j|k-1} C_k^T + R_k \quad \text{(Innovation or residual covariance)}
\]
\[
K_k = P_{j|k-1} C_k^T S_k^{-1} \quad \text{(Optimal Kalman Gain)}
\]
\[
\hat{X}_{k|k} = \hat{X}_{j|k-1} + K_k \tilde{z}_k \quad \text{(Updated Status Estimate)}
\]
\[
P_{j|k} = (I - K_k C_k) P_{j|k-1} \quad \text{(Updated Estimate Covariance)}
\]

The state of the filter is represented by two variables. \( \hat{X}_{j|k} \) represents the estimate of the state at time \( j \) given observations up to and including time \( k \). \( P_{j|k} \) is the error covariance matrix, which is a measure of the estimated accuracy of the state estimate. When we choose \( j < k \), it smoothen the states. When \( j = k \), Kalman Filter is used as a filter only. If \( j > k \), it can perform as a predictor.

Based on [95], for one-step-ahead prediction, the Kalman Filter equations can be transformed into:

\[
\hat{X}_{k+1|k} = A_k \hat{X}_{k|k-1} + B_k u_k + K_k (Y_k - C_k \hat{X}_{k|k-1})
\]
\[
= (A - K_k C) \hat{X}_{k|k-1} + B_k u_k + K_k Y_k \quad \text{(Collecting all equations)}
\]
\[
\hat{X}_{0|1} = \hat{X}_0
\]
\[
K_k = A P_{k|k-1} C^T (C P_{k|k-1} C^T + R)^{-1} \quad \text{(Kalman Gain)}
\]
\[
P_{k+1|k} = A P_{k|k-1} A^T + Q - A P_{k|k-1} C^T (C P_{k|k-1} C^T + R)^{-1} C P_{k|k-1} A^T \quad \text{(Riccati Equation)}
\]
\[
P_{0|1} = P_0
\]

For \( j \)-step-ahead prediction, part of these equations can be transformed into:
\[
\begin{align*}
\hat{X}_{k+1|k} &= A'\hat{X}_{k|k} + \sum_{i=0}^{l-1} A^{i-1}Bu_{k+i} \\
P_{k+1|k} &= A'P_{k|k}(A')' + \sum_{i=0}^{l-1} A'Q(A')
\end{align*}
\]

7.2 Kalman Equations for TCP/AQM

From Figure 3, and equation 4.3, when the network contains a small portion of High-Priority flows, assuming all of the High-Priority flows have sensed congestion and are holding 1 as QoS Alert, we can get \( K_p \approx 0 \). Thus, when split the classic RED controller from \( C_{pc-red}(s) \), we can get the linearized model of TCP/PC-AQM system has \( C(s) \) as controller and \( P(s) \) as its transfer function.

\[
C(s) = C_{pc}(s)
\]

\[
P(s) = \frac{1-K_p}{1-r_p}\left(\frac{2N_p}{R_0^2C} + \frac{K_p}{r_p}\left(\frac{2(1-r_p)N_p}{R_0^2C}\right)\right) - \frac{RC^2}{2N_n^2} - \frac{N_n}{R} - \frac{1}{1-r_p}\left(\frac{2N_p}{R_0^2C}\right) - \frac{2N_n}{R^2C} - \frac{1}{R}
\]

\[
C^2\left(\frac{2(1-r_p)N_p}{R_0^2C}\right) = \frac{N_n}{R} - \frac{1}{R}
\]

It is observed that \( P(s) \) is equivalent to \( P_n(s) \), which is the combined transfer function of Non-Priority flows' TCP window and queue length.

\[
P_n(s) = \frac{R_0C^2}{2N_n^2} - \frac{N_n}{R_0} - \frac{R_0}{\frac{2N_n}{R_0^2C} + \frac{1}{R}}
\]

Thus, assume the delay of the network is a constant \( R \), the input time delay system of TCP/AQM network can be presented using the following equations:
\[
\dot{X}(t) = \begin{bmatrix}
-\frac{2(1-r_p)N}{R^2 C} & 0 \\
\frac{(1-r_p)N}{R} & -\frac{1}{R}
\end{bmatrix}
\begin{bmatrix}
\delta W(t) \\
\delta q(t)
\end{bmatrix}
+ \begin{bmatrix}
-\frac{RC^2}{2N^2(1-r_p)^2} \\
0
\end{bmatrix}
\delta p(t - R)
\]

\[
= AX(t) + Bu(t - R)
\]

\[
Y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \delta W(t) \\
\delta q(t)
\end{bmatrix} = CX(t)
\]

Thus, we have:

\[
X(t) = \begin{bmatrix}
\delta W(t) \\
\delta q(t)
\end{bmatrix}
\]

\[
A = \begin{bmatrix}
-\frac{2(1-r_p)N}{R^2 C} & 0 \\
\frac{(1-r_p)N}{R} & -\frac{1}{R}
\end{bmatrix}
\]

\[
B = \begin{bmatrix}
\frac{RC^2}{2N^2(1-r_p)^2} \\
0
\end{bmatrix}
\]

\[
C = \begin{bmatrix} 1 & 0 \\
0 & 1
\end{bmatrix}
\]

\[
u(t - R) = \delta p(t - R)
\]

In IPv4 network, it is unable for the controller running this model to observe the change of the window size, as the TCP window size is available at the source side instead of the router side. However, in IPv6 network, we can use the header to carry such information. Thus, the router can be informed with all three variables needed, \(\delta W(t)\) \(\delta p(t)\) and \(\delta q(t)\), to predict the next state of the queue size. Of course, considering the limitation of the IPv6 header capacity, it is not recommended for all of the packets to carry the current window size whenever they get transmitted. The congestion affects both sides of the network. TCP sources can sense packet droppings via duplicated Acknowledgement packets. Routers also experience queue sizes which are larger than usual. We can suggest TCP sources to label their outgoing packets with the current window sizes when they sense congestion. But usually when a TCP source is aware of the congestion, it goes back to the initial window size, which is really small. In this case, it is not necessary for the TCP source to warn the router with the size of its next transmission, as its importance is
rather weak compared to the overall work load. Besides the source does not really have a
global supervision of the entire network. Hence, it is more reasonable for the router to
trigger a Global Window-Size-Marking Announcement (GWA) when it senses
congestions.

7.3 Kalman PI Controller Algorithms

The network delays can undermine system stability. If we consider a network
consisting of only Non-Priority flows with network parameters and PI gains as follow,
which is the set of parameters repeatedly used in our previous simulations:

\[ N = 20, \ C = 125 \text{ packets/second}, \ R = 0.2 \text{ second}, \ K_p = 0.102911, \ K_i = 0.051455 \]

We can get the system response in Figure 78 when the reference queue size is set to
be 50.

![Figure 78](image)

**Figure 78** Queue size controlled by PI controller with time delays

However, if the system does not have the time delay, the output is shown in Figure
79, which shows improved performance.
Since it has been clearly observed in most of our previous simulation that the extra Priority List maintenance has brought small oscillation into the queue size due to extra delays caused by the computation, it is necessary for us to find an approach to make up on the over consumed time in PC-AQM algorithms.

Considering the PI TCP/AQM structure a time delay network control system shown in Figure 80, in order to delete the time delay factor $e^{R_s}$, the feedback signal should be taken right after $P(s)$. Our main objective is to use the value observed from point $V_{after}$ to predict the output at point $V_{before}$, and use it as the feedback signal for the PI controller.

The Kalman predictor is used in conjunction with PI controller in the following diagram depicted in Figure 81 to counteract the delay factor.
Chapter 7 Active Queue Management Algorithms with Prediction Mechanism

TCP/AQM Dynamics

TCP Window $\delta W(t)$ AQM Queue

$\delta p(t)$ (u(t)) $\delta q(t)$ $\delta W(t)$ $\delta W(t)$ $\delta q(t)$ $\delta q(t)$ $\delta q(t)$ $\delta q(t)$ $\delta q(t)$

Variable Buffer

$\delta q(t)$ (Y(t))

PI Controller $\delta \tilde{q}(t)$ Kalman Predictor $\delta q(t)$ Status Analyzer

Figure 81 Kalman PI Controller Structure

To minimize computation, a Status Analyzer is used in this structure. It is used to trigger the Global Window-Size-Marking Announcement and the Kalman Predictor sequentially when the change of the queue size exceeds a certain threshold.

The process of GWA is very straight forward. Since all of the TCP-Based protocols react to congestions by analyzing their Acknowledgements, the router uses one bit in the Acknowledgement header, referred as GWA bit acting as a trigger to inform the sources to label their outgoing packets. If TCP source receives an ACK with 1 as GWA bit, it marks all of the packets going out until it receives an ACK carrying a 0 GWA bit.

Kalman Predictor will then observe the current state from the Variable Buffer, which is used to hold all of the information needed for its calculation, and predict any congestion in the future. PI controller uses the output of Kalman Predictor to generate the dropping probability of the incoming packets, thus reduce the probability of large oscillation of the queue size when the network load varies.
Chapter 7 Active Queue Management Algorithms with Prediction Mechanism

7.4 Kalman PI Controller Simulation

We verify our Kalman PI Controller using SIMULINK. A TCP/AQM system using PI as the controller is structured first in Simulation 15. In Simulation 16, a Kalman Predictor is programmed and its accuracy is analyzed. Simulation 17 shows the queue performance under PI AQM controller and Kalman PI controller.

Simulation 15

Referring to the complete stability region for PI-AQM controllers provided in [92], we first set up a network with 60 FTP flows including 6 High-Priority flows, with $C = 3750$ packets/second, $R = 0.246$ seconds, $q_0 = 50$ packets. The gains for P controller and I controller are $K_p = e^{-4}$, $K_i = 6e^{-5}$, respectively. The queue size outputs under PI and KPI controllers are presented as shown in Figure 82.

![Figure 82](image-url)

TCP/AQM with PI and KPI Controller
Following recommendation presented in [92], we now reduce the link capacity to $C = 1250$ packets/second, $R = 0.22$ seconds, $q_0 = 50$ packets. The gains for P controller and I controller are $K_p = 7.5546e-4$, $K_i = 1.4984e-3$, respectively. The queue size outputs under PI and KPI controllers are presented in Figure 83.

![Figure 83: TCP/AQM with PI and KPI Controller](image)

Finally, we increase the number of flows to 75 FTP, and reduce the propagation delay to 0.15 seconds. $K_p$ and $K_i$ are set to be 0.0055 and 0.0023 respectively. The queue size outputs under PI and KPI controllers are presented in Figure 84.
The Kalman predictor is parameterized to predict the queue size based on the linear TCP/AQM model provided in [21]. It can work with not only PI controller, but also traditional RED or the PC-AQM algorithms presented in previous chapters, as $r_p$ could be set as 0 in the model.
Simulations in MATLAB exhibit remarkable improvement when using KPI compared to PI controller.

The digital implementation of KPI can display less perfect result than in SIMULINK, as the computation effort put in prediction and data collection is also time-consuming. However, comprehensively, the reduction of time delays brought by Kalman predictor can not be neutralized. KPI still has its advantages while handling control systems with time delays.
Chapter 8 Recommendations and Conclusion

8.1 Proposed Priority Checking AQM Algorithms

In previous chapters, a set of Active Queue Management algorithms with Priority Checking mechanism are introduced. Their algorithms have been presented and studied. Their performances have been verified via simulations in NS2 and/or MATLAB.

Since Random Early Detection is the AQM algorithm that is most widely accepted and implemented, four PC-AQM algorithms proposed in Chapter 4 and Chapter 5 are RED-Based algorithms. As it is observed from many simulation results that the Proportional Integral Controller can actually achieve better control output of the queue than the RED controller, our next two PC-AQM algorithms are developed from PI controller in Chapter 6. The prediction mechanism presented in Chapter 7 can work with various controllers, such as PI controller or RED controller, whether or not the controller is with or without Priority Checking Mechanism.

![Diagram of PC-AQM Components and Functions]

1. Check the Flow ID and QoS/QoSA
2. Label the QoSA when Applicable
3. Use Calculated Dropping Probability in conjunction with QoS/QoSA to Discard
4. Inform the Status Updater with the Latest Dropping Action

Figure 85 PC-AQM Components and Functions
Chapter 8 Recommendations and Conclusion

The basic working procedure of PC-AQM algorithms is shown in Figure 85. In order to keep a brief reference for the priority checker / labeler, the routers need to hold an extra priority list and enable a maintenance function. In addition, as the packet has carried QoS Requirement and QoS Alert as a record of its QoS status, the router is recommended to be able to provide, apart from the buffer management method with QoS Alert consideration, the related queuing services based on the QoS information the packet carries.

8.2 Network Architecture to enable PC-AQM Algorithms

In a network supporting PC-AQM, first of all, there should be Priority List in each node along the path of the transmission. At the edge of the network, the index of the Priority List should be the flow id. For example, the destination IP address and port number, the source IP address and port number, along with its QoS Requirement, could form a unique pattern of the flow. When the packet gets transmitted into a DiffServ Network, the edge router can take the current QoS Alert into account while aggregating packets for further forwarding behavior. It is also recommended for the DiffServ routers to hold the Priority list to record the current QoS status of each of the aggregated dataflows. An illustration for the above discussion is shown in Figure 86.

The overall structure of the network will be introduced below, followed by the router configuration.

8.2.1 Overall Structure

For an uncomplicated network topology, the transmission does not invoke packet aggregation. Routers simply check the QoS Requirement information carried in the IPv6 header of the packet, refer to the Priority List and mark the QoS Alert field in the header to provide reference to further queue management actions before en-queuing the packet. The overall structure for simple router to router transmission is shown in Figure 87.
Chapter 8 Recommendations and Conclusion

User: Defining QoS Requirements, based on applications used, content accessed, various time slots, or prices willing to pay.

Router: Enabled with Priority Checker and Packet Labeller, holding Priority List and its maintenance function.

Edge/Core Router: Classifying Packets according to their QoS Requirements, metering them based on their current QoS status and aggregating them referring to the QoS Alerts, so as to provide corresponding forwarding behaviours.

Figure 86  Network Components

Figure 87  Router to Router

Queuing techniques with QoS considerations or resource fair share considerations, such as Weighted Fair Queuing and Class-Based Weighted Fair Queuing, can be used in conjunction with the PC-AQM to organize the packets in the buffer, in order to provide some level of QoS support and help to avoid flows without QoS Requirement getting over degraded.

For networks invoking transmission between networks, the classifier at the edge router should take the QoS Alert information into account while aggregate packets. In most of the PC-AQM algorithms, we suggested the packet to hold a either 1 or 0 QoS
Chapter 8 Recommendations and Conclusion

Alert. For GQA-RED, the QoS Alert varies from 0 to 7 indicating different gravity of the QoS crisis that the flow has been through.

![Diagram of network and routers](image)

**Figure 88  Network to Network**

Considering Router A1 and the source nodes connected to it as a network with Star topology, as illustrated in Figure 88, the Router to Router network is actually a Network to Network topology. Thus it is possible to consider Networks B1 and B2 to be two routers handling massive data flows. From a black-box point of view, from the incoming interface of the Edge Router in Networks B1 and B2 to the outgoing interface of the Edge Router on the other side of B1 and B2, the basic inner structure should also consists of the functions that a PC-AQM router carries. The aggregated packets flowing in Networks B1 and B2 should be labeled

### 8.2.2 Router Functionalities

The router functions include Priority Checking, Packet Labeling, Discard Testing and Priority List Maintenance, as depicted in Figure 89.

Working with Fair Queuing-Based queuing techniques, it is not necessary to consider QoS Alert while conducting the queues. However, the Discard Testers are
recommended to check the QoSA fields when generating corresponding dropping probability of certain dataflows.

![Diagram of Router Functions with Fair Queuing-Based Queuing Techniques]

Figure 89 Router Functions with Fair Queuing-Based Queuing Techniques

### 8.3 Conclusion

Nowadays, the number of Internet users is increasing at lightening speed, and there is a mixture of available applications supported by the Internet with varying Quality of Service Requirements.

While the Internet is growing rapidly, the services provided are correctly developing at a sufficient rate. However, users of the same application may wish to pay more for a higher level of QoS. In which case, the same application may contain different levels of value. This possibly has brought a lot of challenges to the future development of the Internet communication. It is very time and money consuming to create a whole new system to take over the exciting enormous global communication system, which allows us to get what we need, and sometimes what we do not need, faster than before. The TCP/IP network is a gigantic system consists of many mechanisms. It is necessary for us...
to develop simple to adopt algorithms to help the users to achieve as much of what they desire as possible.

The QoS demanded primarily depends on two factors, the delay and the throughput. Real-time applications, such as interactive multimedia, are more sensitive to the former factor. Hence their quality is affected more by changes of network delay than by changes of the throughput. Traditional applications, such as FTP, are throughput-sensitive and their quality is almost solely dependent on the throughput that they obtain. The main objective shared by PC-AQM algorithms is to increase the throughput of the dataflows with QoS Requirements, so the QoS Requirements have been transformed into desired average dropping rate of the flow. Both PC-RED and PC-PI based algorithms have helped High-Priority flows to achieve much larger throughputs throughout the simulations.

Reducing the network delay, from a queue management point of view, is keeping the queue size as small as possible so as to reduce queuing delays. In AQA-RED, when adapting router parameters, take maximum dropping probability for example, according to the network variables, such as the ratio of High-Priority flows to overall work load, not only did the queue size decrease to a low value, indicating a decreasing queuing delay, both of the throughput of High-Priority flows and Non-Priority flows increased as well.

This thesis made contributions to priority related QoS in TCP/IPv6 networks summarised as below:

1 QoS-RED and QoSA-RED to bring Priority Checking into existing AQM algorithms. This thesis presents a methodology for the AQM algorithms to work in conjunction with the IPv6 flow labelling ability, to provide acceptable Quality of Service according to the end users' dynamically varying QoS requirements. A simple and generic process, which is adequately adoptable in Differentiated Services (DiffServ) TCP/IPv6 environment, is provided. The simulation results have verified the advantage of our algorithms offered to High-Priority flows in both simple and complex TCP/IPv6 network topologies.
Chapter 8 Recommendations and Conclusion

2 **AQA-RED and GQA-RED to protect Non-Priority flows under congestion.** We have developed a PC-AQM algorithm, AQA-RED, in which AQM parameters are adapted based on the observed ratio of High-Priority flows to overall work load. Thus, AQA-RED can protect the throughputs of Non-Priority flows under heavy congestion without decreasing the QoS status of the High-Priority flows. To avoid the Non-Priority flows becoming unnecessarily overly undermined, we have developed GQA-RED to label the IPv6 header with moderately varying QoS status. Simulation results shown obvious improvement of the throughput of Non-Priority flows under heavy congestion.

3 **PC-PI and DSPCPI to reduce the response time of the Priority-aware TCP/AQM control system.** Compared to PC-RED, PC-PI has a much smaller response time. DSPCPI inherits the stable performance from PC-PI controller and uses double statuses to control the queue size. Since the switch between the two statuses is triggered only when DSPCPI considers the congestion is critical, the computation does not significantly affect the queue performance.

4 **AQM with Prediction mechanism to compensate for the time delays caused by the Priority Checking and Packet Labelling actions.** Considering the PI TCP/AQM structure as a time delay network control system, we use prediction mechanism to use the value observed from the controller side to predict the output of the system in the next sampling time interval, and use it as the feedback signal for the controller. The simulation results have shown remarkable improvement of the system response.

5 **Finally, a recommend network structure to support such PC-AQM algorithms presented above is introduced.** We aim to keep the effort of modification to the existing infrastructure to a minimum. It is theoretically implementable for the routers in DiffServ network to
support PC-AQM algorithms, when they are embedded with some software modification.

This thesis has achieved all of the proposed objectives mentioned in Chapter 1. At last, we hope the works presented in this thesis can help to stimulate related research on TCP/IPv6 network QoS supporting algorithms.
References

References

Distance Networks", Technical Report, Computer Science Department, NC State University, 2003.


References


[57] D.J.Leith, R.N.Shorten, "H-TCP Protocol for High-Speed Long-Distance Networks", 140


[69] C. Brandauer, G. Iannaccone, C. Diot, T. Ziegler, S. Fdida, M. May, "Comparison of Tail Drop and Active Queue Management Performance for Bulk-date and Web-like


[84] NS2: "Network Simulator 2", available online at: http://www.isi.edu/nsnam/ns/


