Parameter Self-Tuning in Internet Congestion Control

by

Wu Chen

Doctoral Thesis

Submitted in partial fulfilment of the requirements
for the award of

Doctor of Philosophy
of
Loughborough University

March 2010

ACKNOWLEDGEMENTS

First I want to express my gratitude to my supervisor, Professor Shuang-Hua Yang, for his kindly guidance and support. He has given me a priceless insight to the state-of-the-art technologies in the networking area. This has given me the opportunity to make great achievements. I also appreciate his valuable comments on my papers and thesis, and his inspiration throughout my study.

My gratitude also extends to Dr Chris Hinde for serving as my Director of Research. I have benefited a lot from his excellent instruction. I have been very fortunate to have had him as my Director of Research.

I appreciate Dr Iain Phillips, Dr Lin Guan, Dr Roger knott, Mr Andre Schappo and Professor David Parish for their support and kindly help during my study. Appreciation is also due to Dr Richard Buxton for offering guidance in statistics.

My PhD study is supported financially in part by Faculty of Science at Loughborough University.

I would like to thank fellow graduate student, in particular the members of our research group for inspiration both during our research group meetings and one-on-one conversations.

Last but not least, I would most like to thank my parents, my sister, brother and other family members for their unconditional support and love. Without their encouragement and support I could not go further in my study.

List of Publications

[1] W. Chen, “Statistical Tuning of RED in Dynamic Network Scenarios,” to be submitted.

[2] W. Chen, “Statistical Tuning of PI Controller in Dynamic Network Scenarios,” to be submitted.

[3] W. Chen and S. H. Yang, “The Mechanism for Adapting RED Parameters to TCP Traffic,” Computer Communications, vol. 32, No. 13-14, pp. 1525-1530, August 2009.

[4] W. Chen and S. H. Yang, “An Algorithm for Adapting RED Parameters to TCP Traffic”, in Proceedings of 2008 IEEE International Conference on Communications, Beijing, China, pp. 5576-5580, May 2008.

[5] W. Chen, Y. Li, and S. H. Yang, “An Average Queue Weight Parameterization in a Network Supporting TCP Flows with RED”, in Proceedings of 2007 IEEE International Conference on Networking, Sensing and Control, London, UK, pp. 590-595, April 2007.

Abstract

Active Queue Management (AQM) aims to achieve high link utilization, low queuing delay and low loss rate in routers. However, it is difficult to adapt AQM parameters to constantly provide desirable transient and steady-state performance under highly dynamic network scenarios. They need to be a trade-off made between queuing delay and utilization. The queue size would become unstable when round-trip time or link capacity increases, or would be unnecessarily large when round-trip time or link capacity decreases. Effective ways of adapting AQM parameters to obtain good performance have remained a critical unsolved problem during the last fifteen years.

This thesis firstly investigates existing AQM algorithms and their performance. Based on a previously developed dynamic model of TCP behaviour and a linear feedback model of TCP/RED, Auto-Parameterization RED (AP-RED) is proposed which unveils the mechanism of adapting RED parameters according to measurable network conditions. Another algorithm of Statistical Tuning RED (ST-RED) is developed for systematically tuning four key RED parameters to control the local stability in response to the detected change in the variance of the queue size. Under variable network scenarios like round-trip time, link capacity and traffic load, no manual parameter configuration is needed. The proposed ST-RED can adjust corresponding parameters rapidly to maintain stable performance and keep queuing delay as low as possible. Thus the sensitivity of RED's performance to different network scenarios is removed. This Statistical Tuning algorithm can be applied to a PI controller for AQM and a Statistical Tuning PI (ST-PI) controller is also developed. The implementation of ST-RED and ST-PI is relatively straightforward. Simulation results demonstrate the feasibility of ST-RED and ST-PI and their capabilities to provide desirable transient and steady-state performance under extensively varying network conditions.

Table of Contents

Table of Figures

Table of Tables

List of Abbreviations

ACK	Acknowledgment
AIMD	Additive Increase Multiplicative Decrease
AQM	Active Queue Management
AP-RED	Auto-Parameterization Random Early Detection
ARED	Adaptive Random Early Detection
AVQ	Adaptive Virtual Queue
CE	Congestion Experienced
DSL	Digital Subscriber Lines
ECN	Explicit Congestion Notification
E-RED	Exponential-Random Early Detection
EWMA	Exponential Weighted Moving Average
FTP	File Transfer Protocol
GMA	Geometric Moving Average
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
ISP	Internet Service Provider

LRED	Loss Ratio-based Random Early Detection
MIMD	Multiplicative Increase Multiplicative Decrease
MSS	Maximum Segment Size
PI	Proportional Integral
QoS	Quality of Service
RED	Random Early Detection
REM	Random Exponential Marking
ST-PI	Statistical Tuning Proportional Intergral
ST-RED	Statistical Tuning Random Early Detection
SYN	Synchronization
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WRED	Weighted Random Early Detection

List of Symbols

B	Buffer size
C	Link capacity
C_{red}	RED control strategy
CongWin	Congestion window size
Count	The number of unmarked packets entering the queue since the last packet was marked
d_{ref}	Target average queuing delay
freeze_time	The minimum time interval between two successive updates of the marking probability
g^2	Detection function
h_l	Coefficient of the upper threshold
h_s	Coefficient of the lower threshold
interval	Time for adjustment of $p_{\max}$
k	Adjustment factor
K_p	Proportional gain
K_{PI}	PI gain
min_{th}	Minimum threshold
max_{th}	Maximum threshold

N	Number of TCP flows
$p_{\max}$	Maximum drop/mark probability
P_{queue}	Queue dynamics
$price$	Price reflecting not only current congestion but also the change in congestion
P_{tcp}	Linearized TCP dynamics
q	Instantaneous queue size
$\widetilde{q}$	Estimated average queue size
q_{ref}	Target queue size
R	Round-trip time
$\overline{R}$	Average round-trip time
RcvWindow	Receive window
RcvBuffer	Receive buffer size
threshold	The queue size above which the marking probability should be updated
T_s	Sampling interval
W	Window size
w_q	Average queue weight
x	Aggregate input rate
z	PI zero
α	Increment
β	Decrease factor
γ	Coefficient of the detection range
δ_1	Increment used in BLUE
δ_2	Decrement used in BLUE
ε	Acceleration factor
ζ	Fraction of the theoretical settling time

σ	Standard deviation
φ	Smoothing parameter
ϕ	Forgetting factor
τ	Desired link utilization
ω_g	Loop's unity gain crossover frequency

CHAPTER 1
Introduction

1.1 Internet Congestion Control

Nowadays more and more people are surfing the Internet as part of their daily lives. They read news on the Web, search information via the Internet, watch video-on-demand whenever they have free time, play online games, and talk with their friends through p2p telephony services. Simultaneously, numerous small networks are linked together and as a result the heterogeneous and huge Internet comes into being. Although circuit-switched networks and packet-switched networks are prevalent in today's telecommunication networks, traditional circuit-switched telephone networks are now also evolving into packet-switched networks. This is because packet-switched networks can provide various communication services as well as reducing the cost of running and maintaining these services.

At the core of the Internet are routers that connect a variety of end systems together via communication links. Thanks to statistical multiplexing in packet-switched networks, i.e. on-demand sharing of resources, end users can share scarce resources such as communication links. Thus high resource utilization can be achieved. However, with only best-effort services provided by the Internet Protocol (IP), it is difficult for various end users to control themselves their sending rate and to prevent over use of the limited resources. Congestion occurs on a communication link when the aggregate

input rate on that link exceeds its capacity. Congestion can potentially lead to severe service degradation. Possible results of congestion include large queuing delay, high packet loss rate, frequent packet retransmission and even congestion collapse when network links are fully utilized but little throughput is useful for the receivers [23][36]. Congestion collapse in the Internet was first observed in the mid 1980s.

To avoid this we need network components (hosts, routers, etc) to work cooperatively. The basic idea behind the Transmission Control Protocol's (TCP's) congestion control mechanism is to adjust the sending rate of a TCP sender according to the congestion level in the network. The congestion level may be measured by packet loss or Explicit Congestion Notification (ECN) [24][25]. ECN allows routers to allocate the Congestion Experienced (CE) codepoint in the header of IP packets as an indication of congestion to the sender [24]. The initial TCP's congestion control mechanism was developed by Jacobson in 1988. After that a number of modifications such as newReno and Vegas were proposed. These end-to-end congestion control mechanisms have been a critical factor in preventing today's Internet from congestion collapse.

Although the robustness of today's Internet is mainly due to TCP's congestion avoidance mechanisms, it is not fully understood if these end-to-end congestion control mechanisms can provide a sufficient service. Since routers have a unified view of traffic passing through them, routers may participate in controlling their own resources and then complement the endpoint congestion avoidance mechanisms. It may be advantageous for a router to drop (or mark the header of) packets before its buffer overflows so that the sender can respond to the upcoming congestion. Thus more severe congestion can be avoided. This proactive mechanism is known as Active Queue Management (AQM). The benefits of AQM include high link utilization, low queuing delay, and low loss rate [8]. Therefore, the deployment of AQM may improve Quality of Service (QoS) of a network. Random Early Detection (RED) was the first AQM mechanism widely deployed in routers [8][21].

1.2 Motivation

A buffer is used to absorb bursty packets during congestion. The traditional scheme for queue management, known as drop-tail, discards incoming packets when the buffer is full. Although it has served the Internet for many years, it results in several significant problems such as high queuing delay, high loss rate, lock-out and global synchronization of flows throttling back, followed by a sustained period of lowered link utilization, reducing overall throughput [8].

By proactively dropping or marking packets before a buffer overflows, Active Queue Management (AQM) aims to stabilize the average queue size at a small value and thus provide both high link utilization and low queuing delay. Additional advantages of AQM include the avoidance of global synchronization and lock-out phenomena [8][41]. In recent years, many AQM algorithms such as Adaptive RED (ARED) [22], Proportional Integral (PI) controller [32][33], Random Exponential Marking (REM) [5], BLUE [19], Exponential-RED (E-RED) [43] and Loss Ratio-based RED (LRED) [57], have been developed to improve the robustness of AQM under varying network conditions. Among them, ARED, PI controller and REM are the most prominent AQM algorithms in the literature [41]. Nowadays RED and RED-like AQMs have been widely deployed in routers [16][69].

It seems promising that the wide deployment of AQM instead of drop-tail would minimize the uncertainty in the Internet [8], but the fact is not so simple. Network conditions in the Internet vary significantly over time. For example, round-trip time may range from the order of milliseconds to over tens of seconds [1]. Available bandwidth for long-lived TCP flows at a particular link may reduce in the face of unresponsive UDP traffic, or decrease when it is divided into several time-varying virtual links to provide different scheduling mechanisms [60]. The stability analysis and experiments in [31][44] demonstrate that a specific parameter setting of RED or PI controller is applicable only under a narrow range of network conditions. The instantaneous queue size would become unstable when round-trip time or link capacity

increases [10][44][60] [63][64]. When bandwidth-delay product decreases the instantaneous queue size would be unnecessarily large [59][63]-[66]. In this situation, the analysis and experiments in [31][33][60] also indicate that the RED or PI controller may become too conservative to provide faster transient response. This is because a higher crossover frequency could be provided to improve responsiveness if RED or PI parameters could be adjusted.

So it is necessary to constantly tune RED or PI parameters to maintain good performance under dynamic network conditions. The authors in [18][22][52][54][61] provided guidelines for adjusting only one of the key RED parameters, i.e., maximum drop probability. The algorithm proposed in [59] explores methods to optimize the RED thresholds by providing tradeoffs with loss rate, but this iteration algorithm needs a long time to achieve the optimal values and thus becomes infeasible in highly dynamic networks. The authors in [60][63]-[66] try to tune several or even all key RED parameters, but these methods depend on the measurement of network parameters. This would increase the overhead of CPU processing in routers or would need additional software and hardware, and thus make the system more complex. In addition, the rough estimation of these network parameters [29][45][60] would increase uncertainty in tuning RED parameters. As to PI controller, the parameter tuning algorithms provided in [11][34][60] can't adjust queuing delay and also depend on the measurement of network parameters. It is still difficult to adapt parameters of RED, PI controller or other AQMs to obtain good performance under dynamic congestion scenarios [13][12][48][57].

As a result, AQMs performs no better than drop-tail when round-trip time varies significantly [4][41], and the queuing delay remains high [16] because conservative AQM parameters have to be used. This is perhaps why drop-tail still has to be investigated and widely used [4][16][40] despite its significant disadvantages. Compare to most propagation delay and transmission delay at access network less than 20ms, most queuing delay is higher than 600ms [16]. This is not what the designers of AQM want because they require that the queuing delay is only a fraction of round-trip time

[22]. Since round-trip time mainly comprises propagation delay, transmission delay and queuing delay, desirable queuing delay should be only the fraction of propagation delay and transmission delay. If every congested link runs AQM with conservative parameters or runs drop-tail with a large buffer size, the Internet suffers from huge round-trip time. As a result, router buffers are still the single largest contributor to uncertainty in the Internet [4].

Considering the significance of congestion control in today's Internet, how to adapt AQM parameters to provide optimal performance under constantly varying network conditions is crucial to the quality of the Internet.

1.3 Contributions of This Thesis

This thesis makes two main contributions. Firstly, the proposed Auto-Parameterization RED (AP-RED) in Chapter 4 unveils the mechanism of adapting RED parameters according to measurable network conditions. Secondly, a Statistical Tuning algorithm is developed to adapt parameters of AQMs according to the characteristics of queue size. Chapter 5 describes the details of the Statistical Tuning RED (ST-RED) and Chapter 6 describes the details of the Statistical Tuning PI (ST-PI). Both ST-RED and ST-PI can maintain desirable performance in a dynamic network. Simulations demonstrate that they can provide better performance than existing AQM approaches.

Chapter 2 surveys existing RED algorithm. It investigates how to set RED parameters and shows how sensitive RED parameters are to varying network conditions. The performance of TCP/RED system has been evaluated and then the effect and reasons for oscillations are analyzed.

Chapter 3 surveys other AQM algorithms. In particular, performances of ARED and Auto-Tuning RED, which can stabilize the average queue size of RED at a reference value, are analyzed. Algorithms and performances of the PI controller, REM, BLUE and Adaptive Virtual Queue (AVQ) are also discussed.

Based on a previously developed dynamic model of TCP behaviour and linear feedback model of TCP/RED, Chapter 4 proposes the mechanism for RED parameter tuning in response to changing network conditions like traffic load, link capacity and round-trip time. This mechanism reveals insights into how four key RED parameters are determined by varying network conditions, and how they can be tuned independently without consideration of the interactions among these RED parameters.

Chapter 5 develops a Statistical Tuning RED (ST-RED) for systematically tuning the four key RED parameters in response to the detected change in the variance of the queue size to control local stability. Under dynamic congestion scenarios, no manual parameter configuration is needed, and ST-RED can adjust RED parameters rapidly to achieve desirable transient and steady-state performance. Thus the sensitivity of RED's performance to different network variables is removed. Simulation results demonstrate that ST-RED provides better performance than existing AQMs when ECN is not enabled.

Chapter 6 applies this Statistical Tuning algorithm to PI controller and also develops a Statistical Tuning PI (ST-PI) controller. The simulation results demonstrate the feasibility of ST-PI and its capabilities of providing desirable transient and steady-state performance under extensively varying round-trip time, bottleneck link capacity, traffic load, Datagram Protocol (UDP) traffic and HTTP sessions. With ECN, ST-PI achieves better performance than existing AQMs.

Finally, Chapter 7 concludes with a summary of research contributions and future works.

CHAPTER 2
RED Literature Review

Because TCP traffic is inherently bursty router buffers are designed to absorb transient bursts without having to drop packets during congestion. However, this results in queuing delay and delay-variance. When buffers overflow packet loss occurs, and when they underflow overall throughput is degraded. Buffers are the main reason behind the uncertainty of end-to-end delay in the Internet [4]. The traditional queue management, drop-tail, has many disadvantages as described in Section 2.1. Because of this RED was proposed by Floyd and Jacobson in 1993 to solve these problems [21]. The RED algorithm is introduced in Section 2.2. However, setting RED parameters is difficult. The choice of RED parameters is discussed in Section 2.3. Then, the performance of RED is investigated and evaluated in Section 2.4. Finally, the deployment of RED is presented in Section 2.5.

2.1 Background

2.1.1 Which links are congested

A link becomes congested when the capacity of the link is saturated by offered load. Congested links may prevent Internet users from making use of the plentiful bandwidth available in other parts of the Internet and may result in significant queuing delay.

In the public Internet, access networks are connected to the rest of the Internet through a tiered hierarchy of Internet Service Providers (ISPs) [38]. Access networks are the first physical links that connect an end system to its edge router, which is the first router on a path from the end system to a remote destination. Tier-1 ISPs are known as Internet backbone networks. Their link capacity is often 622 Mbps or higher. A tier-2 ISP typically has regional or national coverage. In order to reach the global Internet, it connects to a few of tier-1 ISPs. Below the tier-2 ISPs are the lower-tier ISPs, such as tier-3 or tier-4 ISPs, which connect to the larger Internet via one or more tier-2 ISPs. When two ISPs are directly connected to each other, they are said to peer with each other [38].

Most access networks such as Digital Subscriber Lines (DSL) and cable access networks are the bottleneck in the last mile of today's Internet. Tier-2 ISPs or lower-tier ISPs also deploy some congested links which are inside an ISP or are peering links [2][9]. Peering links connect one ISP to other ISPs. Congestion usually occurs during certain limited time periods (e.g., during the morning hours) [42].

On the other hand, most links in tier 1 are not congested so that tier-1 ISPs can provide the best performance. There is a general trend for ISPs to oversubscribe link capacity in order to avoid congestion.

2.1.2 TCP congestion control

TCP controls its sending rate by limiting the number of transmitted but not yet acknowledged packets to a maximum allowable number, referred to as window size W. During a particular round-trip time R, TCP's sending rate is roughly W / R [38].

TCP provides a flow control service to match its sending rate against the reading rate of the receiving application; otherwise the receiver's buffer may overflow. TCP realizes this by having the sender maintaining a variable called the receive window, which is the amount of spare room in the receiver's buffer. The sender's window size is limited to the receive window. The receive window is calculated as follows [38].

$$\text{RcvWindow} = \text{RcvBuffer} - [\text{LastByteRcvd} - \text{LastByteRead}] \quad (2.1)$$

where RcvWindow denotes the receive window, RcvBuffer denotes the receive buffer size the receiver allocates to the connection. LastByteRead denotes the number of the last byte in the data stream read from the buffer by the application process in the receiver. LastByteRcvd denotes the number of the last byte in the data stream that has arrived from the network and has been placed in the receive buffer.

In addition, TCP provides an end-to-end congestion control service to restrict the sending rate in order to avoid network congestion. One cost of congestion is a large queuing delay, when the packet-arrival rate approaches or exceeds the outgoing link capacity. Retransmission may occur in the face of large delays. This wastes the bandwidth by sending unneeded copies of packets. Congestion also results in packet loss when the buffer overflows. In this case, the sender needs to retransmit the lost packets. At the same time, the transmission capacity that was used at each of the upstream links to forward that packet to the point at which it was dropped, ends up having been wasted [38].

Network congestion can be indicated by timeout event, three duplicate ACKs, increasing round-trip time [38] or a set CE codepoint [24][25] in the IP header. The rationale behind congestion control is for the TCP sender to reduce its sending rate in the face of detected congestion. The sender controls its sending rate by setting its congestion window size, denoted by CongWin.

When a TCP connection begins, the value of CongWin is typically set to 1 Maximum Segment size (MSS), resulting in an initial sending rate of roughly MSS/R. Then the TCP sender increases its rate exponentially by increasing the value of CongWin by 1 MSS for each acknowledged segment. In this initial phase, which is called Slow Start, CongWin doubles every R. After CongWin reaches a threshold, CongWin grows linearly by increasing CongWin by 1 MSS every R. This linear increase phase of TCP's congestion control protocol is known as congestion avoidance. If a timeout event is detected, TCP cuts CongWin to 1 MSS and start Slow Start again. If a triple

duplicate acknowledgment (ACK) is received, TCP Reno halves the current value of CongWin and starts congestion avoidance, while TCP Tahoe responds the same as the presence of a timeout event and enters into Slow Start. The threshold is initially set to a large value (65 Kbytes in practice) so that it has no initial effect. After the congestion is detected, the value of threshold is set to one half of the current value of CongWin [3][38].

In general a TCP sender additively increases its rate when it perceives that the end-to-end path is congestion-free, and multiplicatively decreases its rate when it detects that the path is congested. For this reason, TCP congestion control is often referred to as an Additive Increase Multiplicative Decrease (AIMD) algorithm [38]. This AIMD algorithm gives a saw-tooth pattern in a long-lived TCP connection as follows:

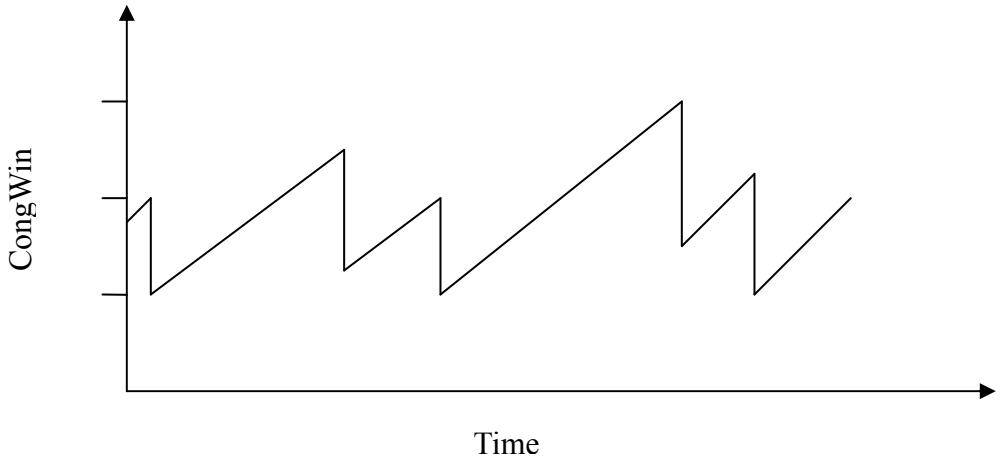


Figure 2.1 AIMD congestion control

Assume that *R* and *W* are constant over the duration of the connection. If *W* denotes the value of CongWin when a loss event occurs, the TCP transmission rate ranges from $W / (2 \cdot R)$ to W/R [47][50]. Thus, each cycle must be $W/2$ round-trip time, or $R * W/2$ seconds.

In general, a TCP sender controls its sending rate according to the receive window and congestion window as follows.

$$\text{LastByteSent} - \text{LastByteAcked} \leq \min\{\text{RcvWindow}, \text{CongWin}\} \quad (2.2)$$

where LastByteSent – LastByteAcked is the amount of unacknowledged data that the sender has sent into the connection.

If the receive window becomes zero, the TCP specification requires the sender to continue to send segments of one data byte.

Assuming that two connections have the same *MSS* and *R*, TCP congestion control converges to provide an equal share of a bottleneck link's bandwidth among these two competing TCP connections. On the other hand, when multiple connections share a common bottleneck, those sessions with a smaller *R* are able to obtain the available bandwidth at that link more quickly (that is, open their congestion windows faster) and thus will enjoy higher throughput than those connections with large *R*. When an application uses multiple parallel connections, it can also grab a larger fraction of the bandwidth in a congested link [39].

It may be unfair for TCP users when they share links with UDP applications. This is because UDP applications do not cooperate with the other connections nor adjust their sending rates appropriately. However, a TCP sender has to reduce its sending rate in the case of increasing congestion (loss). It is possible for UDP flows to crowd out TCP traffic [38].

2.1.3 Traditional queue management

TCP adjusts its sending rate in response to a loss event. A queue management scheme complements TCP by determining a packet-drop policy. The traditional queue management policy is known as drop-tail. It accepts packets until a queue becomes full, and then drops subsequent incoming packets until a packet in the full queue has been transmitted.

Although drop-tail has served the Internet for years, it has several significant problems [8]. One phenomenon is lock-out, when drop-tail allows a single or a few

connections to monopolize the queue, preventing other connections from getting room in the queue. Another problem is the full queue. Since drop-tail notifies congestion only when the queue has become full, the queue has to maintain a full (or almost full) status for long periods of time. When incoming bursty packets from different connections arrive at the full queue, they have to be dropped. This may result in a high loss rate and lead to a global synchronization [8].

Sizing buffers in routers is important for the Internet. Because data traffic is inherently bursty, buffers need to be large enough to absorb this burstiness. However, large buffers may lead to unacceptable queuing delay, while small buffers may lead to excessive packet loss and degrade throughput. The "optimal" buffer size for a router is a function of the relative trade-off between low queuing delay and high link utilization. In practice buffer size should reflect the size of bursts that need to be absorbed. The key to sizing the buffer is to make sure that, while the sender pauses, the router buffer doesn't become empty and force the bottleneck link to go idle. For many years the rule of thumb for sizing the buffer was that the buffer size (B) should be the product of the average round-trip time ($\overline{R}$) and the link capacity (C) [56]. However, recent theoretical and experimental efforts suggest that this rule of thumb is true only for a relatively small number of TCP flows. For a large number of TCP flows (N) typically passing through backbone router links, the buffer size needed is $B = \overline{R} \cdot C / \sqrt{N}$ for long-lived and short-lived TCP flows [4]. Because short-lived TCP flows require very small buffers, the buffer size usually depends upon the number of long-lived flows. However, although the link can be fully utilized, a study has shown that this small buffer size comes at the cost of a high loss rate [15]. Short-lived flows and long-lived flows in this thesis are defined as flows that are in slow-start and in congestion avoidance respectively. This means that flows may transit from short to long during their existence.

2.2 The RED Algorithm

The main objectives of RED are to provide both low average queuing delay and high throughput. Additional objectives include the avoidance of a lock-out phenomenon, of a global synchronization, of a bias against bursty traffic, and the reduction of loss rate [21].

- Estimation of average queue size

RED uses a low-pass filter with an Exponential Weighted Moving Average (EWMA) to calculate the average queue size. For every incoming packet, RED computes the average queue size as follows:

$$\tilde{q} = (1 - w_q)\tilde{q} + w_q q \tag{2.3}$$

where q is the instantaneous queue size, $\tilde{q}$ is the estimated average queue size, and w_q is the average queue weight which determines the time constant of the low-pass filter and determines the degree of burstiness allowed in a queue.

RED provides an option to measure q in byte mode or in packet mode.

- Packet mark (drop) decision

In gentle_ mode [20], the initial drop probability can be calculated according to the average queue size as follows:

$$p_i = \begin{cases} 0 & 0 \le \tilde{q} < \min_{th} \\ \dfrac{\tilde{q} - \min_{th}}{\max_{th} - \min_{th}} p_{\max} & \min_{th} \le \tilde{q} \le \max_{th} \\ \dfrac{\tilde{q} - \max_{th}}{\max_{th}} \cdot (1 - p_{\max}) + p_{\max} & \max_{th} < \tilde{q} < 2\max_{th} \\ 1 & \tilde{q} \ge 2\max_{th} \end{cases} \tag{2.4}$$

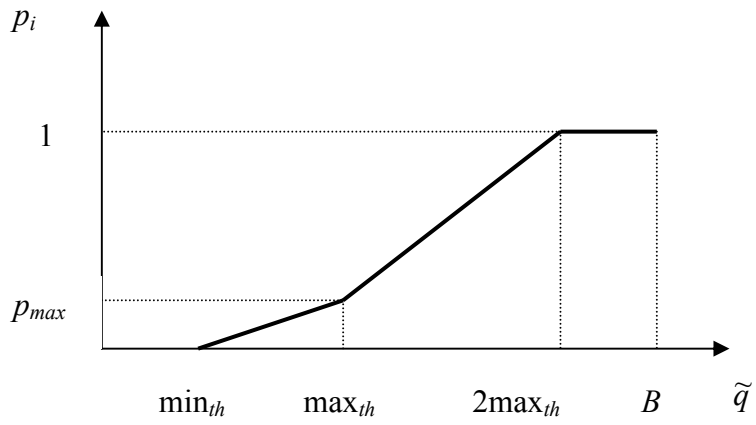


Figure 2.2 RED Drop function with "gentle_" mode

where min_{th} is the minimum threshold that specifies the average queue size below which no packet will be dropped, max_{th} is the maximum threshold, and $p_{\max}$ is the maximum drop probability achieved when the average queue size reaches max_{th}. In the original RED, p_i directly jumps to 1 when the average queue size is greater than max_{th}. The gentle_ modification of RED is suggested to remove the instability related to the discontinuity of the drop function.

The final marking probability for each arriving packet is $p = p_i /(1 - count * p_i)$, where *count* is the number of unmarked packets entering the queue since the last packet was marked. So the intermarking time, the number of packets between two consecutive marked packets, is a uniform random variable.

The final marking probability can be calculated in byte mode or in packet mode. In byte mode, the probability of marking a packet is proportional to its packet size in bytes. The final marking probability in byte mode is calculated as follows.

$$p_i \leftarrow p_i \frac{\text{arriving packet size}}{\text{average packet size}} \tag{2.5}$$

$$p \leftarrow p_i /(1 - count \cdot p_i) \tag{2.6}$$

2.3 Setting RED Parameters

RED's effectiveness is highly dependent upon its operating parameters. In fact, in cases where these parameters do not match network conditions, the performance of the RED gateway can be worse than that of a traditional drop-tail gateway [13][12][48].

Although RED can certainly outperform traditional drop-tail queues, the correct parameters of RED are needed to perform well under different congestion scenarios.

2.3.1 Minimum threshold and maximum threshold

An optimal value of min_{th} should be high enough to accommodate bursty traffic. For typical traffic with large bandwidth-delay product passing through a link, a minimum threshold of one packet would result in unacceptably low link utilization. On the other hand, the maximum value of max_{th} is limited by the maximum average delay that is allowed in a router. In the meantime the space between max_{th} and min_{th} should be large enough to avoid global synchronization. If max_{th} -min_{th} is too small to accommodate the typical increase in the average queue size during a round-trip time, the computed average queue size can regularly oscillate up to max_{th}; this behaviour is similar to the oscillations of the queue up to the buffer size with drop-tail routers. A rule-of-thumb is to set max_{th} to be three times min_{th} [26].

So the optimal setting for min_{th} depends partly on the link capacity, round-trip time, and max_{th}. Similarly, the optimal setting for max_{th} also depends partly on the link capacity, round-trip time, and min_{th}. The optimal values of min_{th} and max_{th} should provide a desired trade-off between low average delay and high link utilization in a router. One rule-of-thumb for the trade-off is that the resulting average queuing delay at a router is a fraction of the end-to-end round-trip time [22].

2.3.2 Maximum drop/mark probability

The mark/drop probability reaches its maximum value $p_{\max}$ when the average queue size reaches the maximum threshold. $p_{\max}$ directly impacts upon the aggressiveness of the RED mechanism. When $p_{\max}$ is configured to be too conservative, the RED queue can degrade into a simple drop-tail queue because it does not notify a sufficient number of sources of the congestion. On the other hand, an aggressive configuration may result in underutilization when many sources cut their sending rates in response to the observed congestion. $p_{\max}$ determines the position of the average queue size among min_{th} and max_{th}. The correct value of $p_{\max}$ depends on network conditions such as traffic loads, link capacity and round-trip time [26].

One recommended value for $p_{\max}$ is 0.1. If a router is operating with steady-state packet drop rates of 20-30% there is something wrong in the engineering of the network [26].

2.3.3 Sampling interval

A RED router updates its average queue size on each packet's arrival, and hence the sampling interval T_s is equal to $1/C$ for a stable queue. C is the arriving rate in packets/second, determined by the average packet size in the network for a constant link capacity (bytes/sec).

2.3.4 Average queue weight

A RED router uses a low-pass filter to calculate the estimated average queue size, and the time constant of the low-pass filter depends on the average queue weight w_q. w_q determines the degree of burstiness allowed in the router. If w_q is set too large, the averaging procedure will not filter out transient congestion at the router. If w_q is set too small, then the estimated average queue size will respond too slowly to reflect

changes in the instantaneous queue size. In this case, the router is unable to detect initial stages of congestion [22].

A guideline for setting w_q is that the estimated average queue size can reflect the congestion lasting longer than a typical round-trip time and then the router can give feedback to the sources. For congestions that last less than a typical round-trip time, the desirable behaviour of the queue is to absorb the congestion without dropping packets. Therefore, the time constant should be at the order of round-trip times, rather than fractions of round-trip times. As the time constant of the filter is $-\frac{1}{C \cdot \log_e(1-w_q)}$, w_q can be set to $w_q = 1 - \exp(-1/nRC)$. Thus the time constant of the estimator is n times round-trip time [22][52]. n = 10 is recommended [52].

2.3.5 Byte and packet mode

The instantaneous queue size can be measured in byte mode or in packet mode. For a queue with capacity in units of bytes, RED should measure the queue in byte mode so as to give an accurate indication of the congestion level. It is meaningless for the RED router to measure the queue in packets because this measurement does not reflect the actual usage of the queue. It is possible for a nearly empty queue to accommodate a large number of small packets or for a full queue to accommodate a few large packets. For a similar reason, RED should measure the queue in packet mode for a queue with capacity in units of packets. A router where the transmission delay for a packet is largely a function of the packet size in bytes usually uses a queue with capacity in units of bytes. For a router where the transmission delay is a constant for any packet it usually uses a queue with capacity in units of packets [27][28].

In addition, RED needs to choose between byte mode and packet mode in relation to its marking policy. When marking in byte mode, RED marks large packets with large probabilities. On the other hand, RED marks any packet with the same probability when operating in packet mode. The choice of the mark mode depends on what the

scarce resource is in the network and the end-to-end congestion control mechanisms. For an end-to-end congestion control protocol such as TCP, an indication of congestion comes from one or more dropped packets from the most recent window of data. In this case, if the scarce resource is link capacity in bytes per second, byte marking mode would be a better choice. The reason is as follows: two arriving traffic with the same rate in bytes per second but different packet sizes would have the same marking probability. However, when RED operates in packet marking mode, the flow with smaller packets will have more packets to transmit and then more packets will be marked. Then the flow with larger packets will gain more bandwidth and there is a strong bias against small packets. In addition, because in byte-marking mode the marking probability for a particular flow is proportional to its bandwidth share in the bottleneck link, this prevents various bandwidth flows from gaining a disproportionate share of the link during congestion and thus provides fairness. On the other hand, RED should drop packets in packet mode if the scarce resource is CPU processing in packets per second.

For a packet to be transmitted in a network, it is possible for it to enter queues with capacity in bytes and to pass through bottleneck links with capacity in bytes per second. So most RED routers measure the queue size in bytes and then marking a packet in bytes.

For the most common network conditions where the queue has its capacity in bytes and the scarce resource is the link capacity, a comparison using two main metrics among different measurements and marking modes were carried out in [14][17]. Based on numerous experiments using different packet sizes, traffic types, and traffic loads, results demonstrate that the choice of marking mode is extremely important for fairness. Although packet marking mode provides somewhat better link utilization than byte marking mode, byte marking mode provides much more fairness under various scenarios [14][17]. The fairness in [17] is defined as:

$$f(x_1, x_2, ..., x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \cdot \sum_{i=1}^{n} x_i^2} \leq 1 \qquad (2.7)$$

where x_i is the total number of bytes transmitted by host *i* and *n* is the total number of hosts. A fairness index of 1 indicates that each host transmits the exact same number of bytes. As for measurement, sometimes the packet mode have better link utilization than byte mode, and sometimes byte mode have better link utilization than packet mode. Their performance depends on the actual packet size and the mean packet size. On the other hand, the queuing delay fluctuate less or has less jitter in byte mode than in packet mode for a normal link capacity in units of bytes. In summary, byte mode measurement and byte mode marking is the best choice in the most common routers.

2.3.6 Mean packet size

When operating in byte mode, a RED router needs to set mean packet size to represent the typical packet size on the link so that incoming packets can be normalized by it. Thus, RED can measure the queue size in a unit of packet numbers instead of a unit of bytes. However, because there are many packets with varying packet sizes passing through a link and the distribution of packet sizes is highly dynamic, it is inaccurate to use a static mean packet size to reflect actual average packet size over time on the link.

It is observed in [17] that RED in both byte measuring and byte marking mode may achieve desirable fairness and relatively stable queuing delay with suboptimal mean packet size. In this case the high utilization could be maintained when the actual mean packet size fluctuates within ±250 bytes of mean packet size. Because the majority of the distribution of average packet sizes falls within a range of 500 bytes, a trial-and-error mean packet size on a link could provide good performance in most cases.

2.4 Evaluation of TCP/RED

2.4.1 Advantages of RED

Rather than wait for full buffers to drop packets, RED allows routers to control when and how to drop packets before buffers overflow. In contrast to drop-tail, RED mechanisms have the following advantages for responsive flows [8].

- Reducing queuing delay

  RED can reduce the queuing delays by keeping the average queue size small.

- Providing high link utilization

  By controlling the average queue size within a given range, the RED router avoids overflow of the buffer or an empty buffer. Because there are always packets to transmit RED maximizes the link capacity.

- Reducing loss rate

  A drop-tail queue will drop bursty packets when the buffer space is insufficient. It is more difficult for TCP to recover from a burst of packets than from a single packet drop and unnecessary dropped packets lead to the potential waste of bandwidth on the way to the drop point. RED keeps the average queue size small and thus has more room to absorb naturally-occurring bursts without dropping extra packets simultaneously. This is significantly borne out by empirical data.

- Avoiding full queue

  RED addresses the full queue problem by dropping packets before the queue overflows.

- Avoiding lock-out behaviour

  RED solves the lock-out problem by constantly providing enough buffers for arriving packets.

- Avoiding global synchronization

By marking packets randomly before the occurrence of a full queue, RED routers avoid marking too many packets too close together. This avoids dropping packets from many connections at the same time and thus avoids global synchronization.

- Avoiding a bias against bursty traffic

There is always room left in the queue to absorb bursty traffic. In addition, since RED marks an arriving packet randomly, this reduces the probability that successive packets from the same bursty traffic are marked. So the probability of marking a packet from a particular connection is roughly proportional to that connection's bandwidth share through the router. All these avoid a bias against bursty traffic.

2.4.2 Weakness of RED

- Sluggish response to a change in instantaneous queue size

RED uses a low-pass filter to average the instantaneous queue size. A large time constant of the low-pass filter has to be used to stabilize the queue so that the system becomes sluggish to reflect the change in the instantaneous queue size [33].

- Stability affected by users

The sampling frequency of RED is determined by the link capacity and packet size so that users can use very small or very large packets to influence the stability of TCP/RED systems [46].

- Global synchronization under highly dynamic network conditions

When there are significant changes in network conditions which make the TCP/RED system unstable, the corresponding marking probability of RED changes significantly over a short time. In this case, RED fails to mark packets evenly over times and therefore global synchronization occurs among sources [19].

- Parameter sensitivity to network scenario

Analysis in section 2.3 has demonstrated that RED parameters are quite sensitive to network scenarios. In particular, the minimum threshold min_{th}, maximum threshold max_{th}, and maximum drop probability $p_{\max}$ are sensitive to round-trip time, link capacity and traffic loads. When network scenario changes, TCP/RED may move into an unstable state and perform worse than drop-tail.

2.4.3 Performance of RED with specific traffic

Considering heterogeneous network traffic, the following performances of RED are compared to drop-tail.

- Web traffic

The Internet is dominated by web traffic characteristic of short-lived HTTP flows. Most Web objects have a small size of 10-20KB and the average Web document is only around 30KB [62]. For these short-lived connections an end-to-end response time is more important than the network-centric measurement [13][12].

At a congested link with traffic loads between 90% and 100% of the link capacity, RED can be carefully tuned to provide better end-to-end response time than drop-tail. However, there exists a complex trade-off between the RED parameters that improve response time for short-lived connections and those that improve response time for longer connections. In addition, RED parameters that provide the best link utilization produce poorer response times [13][12]. It is very difficult to choose the correct RED parameters with which RED can outperform drop-tail.

- Mix of short-lived and long-lived TCP connections

When competing with long-lived TCP connections, short-lived TCP connections under RED can obtain higher goodput than those under drop-tail [62]. On the other hand, long-lived TCP connections under RED tend to have lower goodput than those under drop-tail. Goodput is the ratio of the total number of packets (excluding retransmissions) received by the receivers per unit time to link capacity.

This is desirable for internet users who are sensitive to the delay of interactive applications, as short-lived TCP connections typically belong to interactive applications like the web.

- Short-lived and long-lived TCP connections with different round-trip times

For short-lived connections, the goodput ratio is inversely proportional to the round-trip time ratio no matter under RED or under drop-tail. This is mainly because short-lived connections are dominated by the slow start procedure. Unless the loss rates for these connections are very different, they tend to require similar number of round-trip times [62].

On the other hand, RED tends to provide fair goodput among long-lived TCP connections with different round-trip times, while drop-tail tends to provide higher goodput for those connections with shorter round-trip time [62].

- Mix of TCP and UDP traffic

In general, RED without ECN for TCP flows can increase the loss rate for smooth UDP traffic, and reduce the loss rate for bursty UDP traffic. This is because RED can alleviate bias against bursty traffic by distributing losses uniformly among all connections. On the other hand, RED with ECN always significantly increases the loss rate for all kinds of UDP traffic. This is because TCP becomes more aggressive with the use of ECN [62].

2.4.4 The effect of TCP/RED oscillations

Because of TCP's AIMD and the feedback nature of the TCP/RED system, oscillations in the queue size are very common. Some oscillations are "benign" when they do not significantly affect throughput and delay. Some oscillations are "malignant" in that they lead to periods of high packet marking rates and periods of no packets being marked. This may degrade throughput or increase delay jitter [22]. In addition, when the marking probability of RED varies dramatically over very short periods of time, it

becomes impossible for RED to mark packets randomly and evenly over time. In this case RED fails to avoid global synchronization [19].

2.4.5 Reasons for TCP/RED oscillations

Queue size reflects the mismatch between the aggregate input rate and the link capacity. As a result, the actual queue size in a router depends on the average window at the senders. Although there are obvious oscillations of an individual window due to TCP's AIMD, the individual window's effect on the queue can be smoothed out by the aggregate windows sending packets through the link. As the round-trip time or the link capacity increases, the individual window sizes will vary within a larger range and eventually result in oscillations of the queue. For the same reason a smaller traffic load has a larger average window and may make the queue instable. On the other hand, noise-like mice traffic and heterogeneous delays have little impact on the oscillation of the queue [44].

2.5 RED Implementations

RED is not a default configuration on routers. It can be enabled when there are congested links on a router to reduce queuing delay without degradation of link utilization. Since typical tier-1 ISPs don't run congested links, it is unnecessary for most of them to turn on RED on their routers. RED is suggested to run for a router with oversubscribed links. Cisco suggests that WRED (weighted Random Early Detection) should be used in core routers rather than in edge routers. "Edge routers assign IP Precedences to packets as they enter the network. WRED uses these precedences to determine how to treat different types of traffic.[69]" In addition, access networks like DSL show considerable deployment of RED-style policies [16].

CHAPTER 3
Other AQM Mechanisms Review

3.1 Introduction

As described in Chapter 2, RED may result in bad performance when it is improperly configured. Parameterization of RED is very difficult because RED parameters are tightly coupled to each other and coupled with network scenarios. In order to solve the problem, other AQM algorithms have been proposed in recent years [5][19][22] [33][37][49][52][53][57]. They try to provide robust parameters under varying network conditions or to achieve better performance such as lower queuing delay than the original RED. Among these AQMs, ARED [22], Auto-Tuning RED [52], PI controller [32][33], REM [5], BLUE [19] and AVQ [37] are reviewed in this Chapter.

3.2 Adaptive RED

3.2.1 Objectives

The correct parameters of RED depend on network scenarios like traffic loads, and the performance of RED is quite sensitive to its parameters and to the network scenarios. In order to stabilize the queue in a specified target range and provide high throughput, a RED router needs to update RED parameters in response to changed network scenarios. Adaptive RED (ARED) aims to adapt RED parameters $p_{\max}$ to

varying traffic conditions so that the average queue size is maintained at a reference value [22].

3.2.2 Algorithm

The guidelines for ARED are to adapt $p_{\max}$ to keep the average queue size half way between min_{th} and max_{th} [22]. The algorithm is given as follows.

Every interval:

if ($\tilde{q}$ > target and $p_{\max} \le 0.5$)

$p_{\max} \leftarrow p_{\max} + \alpha$;

if ($\tilde{q}$ < target and $p_{\max} \ge 0.01$)

$p_{\max} \leftarrow p_{\max} * \beta$;

Fixed parameters:

interval: time for adjustment of $p_{\max}$; 0.5 seconds

target: target for $\tilde{q}$; $[\min_{th} + 0.4 * (\max_{th} - \min_{th}), \min_{th} + 0.6 * (\max_{th} - \min_{th})]$

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

β: decrease factor; 0.9

Figure 3.1 The ARED algorithm

The AIMD scheme to adapt $p_{\max}$ is used instead of multiplicative increase multiplicative decrease (MIMD) in [18] and other controls. Experiments demonstrate that AIMD is more robust than other schemes.

A long interval is used so that the dynamics of ARED is dominated by the small time scales of the original RED rather than by frequent adjustment of $p_{\max}$. This slow and infrequent adaptation of $p_{\max}$ is vital for the robustness of ARED [22].

To ensure the performance of ARED is acceptable during transition periods, $p_{\max}$ is restricted within the range [0.01, 0.5]. Considering the uncertainty of the average queue size after a sharp change in network conditions, this restriction prevents ARED from over-tuning $p_{\max}$ during transient period. The upper bound is set to 0.5 because it is an engineering problem to have steady-state drop probability greater than 25% [22]. In this case, the problem needs to be solved from an engineering aspect such as reducing the number of users passing through the congested link. On the other hand, when steady-state drop probability is less than 0.005, RED with the lower bound of $p_{\max}$ set to 0.01 can still run robustly and the actual delay is lower than the target delay [22].

The setting of α to 0.1 ensures that a one step increase of $p_{\max}$ does not move the average queue size from above the target range to below it, and the setting of β to 0.9 ensures that a one step reduction of $p_{\max}$ does not move the average queue size from below the target range to above it [22].

3.2.3 Initialization of RED parameters

In automatic mode max_{th} is set to three times min_{th}, and the average queue size is maintained around 2* min_{th}. As a result, min_{th} is determined by the desired average queue size that reflects the trade-off between throughput and queuing delay. One rule-of-thumb for a plausible trade-off is that the resulting average queuing delay at a router is a fraction of the end-to-end round-trip time. Given a target average queuing delay of d_{ref} seconds and min_{th} no less than 5 packets, in automatic mode min_{th} is set to Max [5, $\frac{d_{ref}C}{2}$] packets, where C is the link capacity in packets/second [22].

In automatic mode, w_q is set to keep the time constant of the estimator for the average queue size on the order of round-trip times. Setting $w_q = 1 - \exp(-1/C)$ gives a time constant ten times the default round-trip time of 100ms [22].

3.2.4 Discussions

ARED can adapt RED parameter $p_{\max}$ automatically in response to varying traffic conditions and could keep the average queue size within the target zone. This provides a simple mechanism to remove parameter sensitivity from unknown network scenarios. On the other hand, it is observed that ARED in byte-mode significantly outperforms that in packet-model [41]. However, ARED uses default network parameters such as a default round-trip time to determine some default ARED parameters such as w_q and interval. In fact these network parameters vary significantly over time. So ARED cannot absolutely remove all sensitivity of RED parameters from network scenarios. Its performance may degrade under some widely changed network conditions such as round-trip time.

3.3 Auto-Tuning RED

3.3.1 Objectives

The main objective of Auto-tuning RED [52] is to keep the average queue size at a reference value despite varying network conditions. This mechanism also gives guidelines for designing other RED parameters.

3.3.2 Choice of fixed RED parameters

The difference between max_{th} and min_{th} can be expressed as a fraction v of the bandwidth-delay product as follows.

$$\max_{th} - \min_{th} = v \cdot R \cdot C$$

And the time constant of the RED estimator can be expressed as a multiple *n* of the round-trip time *R* by setting

$$w_q = 1 - \exp(-1/nRC).$$

Based on control theory analysis, a large *v* is needed for a small *n* to get a phase margin of at least 45 deg and to stabilize the queue. However, a large *v* means large queuing delay and large delay jitter. On the other hand, if *n* is too large, the system response may become slow. Analysis indicates that $n = 10$ is a good choice to obtain both an acceptable *v* and acceptable responsiveness of the system [52].

When choosing $n = 10$, the minimum satisfactory value of *v* is 0.21, that is,

$$\max_{th} - \min_{th} \geq 0.21 \cdot C \cdot R$$

Follow a common guideline $\max_{th} = 3 \cdot \min_{th}$. $\min_{th}$ is set to

$$\min_{th} \geq 0.13 \cdot R \cdot C$$

3.3.3 The reason for adapting $p_{\max}$

Consider *N* TCP connections with the same round-trip time share a link. Each connection receives fair share of *C*/*N* from the available bandwidth *C* in the long run. Thus the window size of each connection is $W = \frac{C}{N} R$.

According to the well-known equation $W = \sqrt{\frac{8}{3}} \cdot \frac{1}{\sqrt{p}}$, *p* can be set to

$$p = \frac{8}{3}\left(\frac{N}{RC}\right)^2 \tag{3.1}$$

At the target queue size $q_{ref} = \frac{\min_{th} + \max_{th}}{2}$, *p* is

$$p = p_{\max} / 2 \tag{3.2}$$

According to (3.1) and (3.2), $p_{\max}$ is obtained as

$$p_{\max} = \frac{16}{3}\left(\frac{N}{RC}\right)^2$$

This demonstrates that $p_{\max}$ is a function of network scenario parameters *N*, *R* and *C* [52]. So $p_{\max}$ need to be adapted in response to varying network scenarios to stabilize the average queue size within the target range.

3.3.4 Algorithm

From equation (3.2), the correct $p_{\max}$ can be derived corresponding to the steady-state value of current drop probability [52] as follows.

$$p_{\max} \leftarrow 2 \cdot (\frac{\widetilde{q} - \min_{th}}{\max_{th} - \min_{th}}) \cdot p_{\max}$$

Thus correct $p_{\max}$ can be obtained by increasing current $p_{\max}$ by the amount:

$$\Delta p_{\max} \leftarrow 2 \cdot (\frac{\widetilde{q} - \min_{th}}{\max_{th} - \min_{th}}) \cdot p_{\max} - p_{\max} \tag{3.3}$$

or by a simplified increment:

$$\Delta p_{\max} \leftarrow 2 \cdot (\frac{\widetilde{q} - q_{ref}}{\max_{th} - \min_{th}}) \cdot p_{\max} \tag{3.4}$$

Equation (3.4) implies that correct $p_{\max}$ can be achieved in a single iteration according to a steady-state value of current $p_{\max}$.

As the theoretical settling time of 40*R* is too long, a small update interval is used by taking a fraction ζ of the theoretical settling time [52]. Then an acceleration factor ε is used to speed up convergence. Therefore, the equation (3.4) is revised to:

$$p_{\max} \leftarrow p_{\max} + \varepsilon \cdot \zeta \cdot \Delta p_{\max} \tag{3.5}$$

Here $p_{\max}$ is restricted arbitrarily within the range [0.01, 0.5] to prevent $p_{\max}$ from becoming too large or too small during transient period. Then the auto-tuning algorithm [52] is described in Figure 3.2.

Every interval:

if ($\widetilde{q} > q_{ref}$ and $p_{\max} < 0.5$)

increase $p_{\max}$:

$p_{\max} \leftarrow Min(0.5, p_{\max} + \varepsilon \cdot \zeta \cdot \Delta p_{\max})$;

if ($\widetilde{q} < q_{ref}$ and $p_{\max} > 0.01$)

decrease $p_{\max}$:

$p_{\max} \leftarrow Max(0.01, p_{\max} + \varepsilon \cdot \zeta \cdot \Delta p_{\max})$;

Fixed parameters:

interval – time for adjustment of $p_{\max}$; 4*R*

ε – acceleration factor; 1.5

ζ – fraction of the theoretical settling time; 0.1

Figure 3.2 Auto-Tuning RED algorithm

3.3.5 Discussions

Auto-Tuning RED can achieve correct $p_{\max}$ from current $p_{\max}$ and current average queue size by a few steps of adjustment. In addition, experiments demonstrate that Auto-Tuning RED without limitation range [0.01, 0.5] of $p_{\max}$ still works well. As a fixed max_{th} and min_{th} are used, the queue may become unstable in the case when *R* or

C is large, or may become unnecessary large in the case when R or C is small. More parameter optimization is needed for this algorithm.

3.4 PI Controller

3.4.1 Objectives

RED is sensitive to its parameters as well as network conditions such as traffic load, link capacity and round-trip time. This motivates the PI controller [32][33] to present a scheme more robust to varying network scenarios.

3.4.2 Transfer function and digital implementation

A PI controller can improve the response time by removing the low-pass filter of RED that results in sluggish response of the system. In addition, the integral part can eliminate the steady-state error and make the queue converge to its reference value. A PI controller has the transfer function [32][33] as

$$C_{PI}(s) = K_{PI}\frac{s/z+1}{s} \tag{3.6}$$

Then the open-loop transfer function of TCP/PI system is

$$L(s) = \frac{\frac{K_{PI}C^2}{2N}(\frac{s}{z}+1)e^{-sR}}{s(s+\frac{2N}{R^2C})(s+\frac{1}{R})}.$$

The design of a PI controller involves choosing the location of the zero z and the PI gain K_{PI}.

Zero z can be chosen to coincide with the corner frequency of the TCP window dynamic as

$$z = \frac{2N}{R^2C}$$

and the loop's unity gain crossover frequency can be chosen as

$$\omega_g = \frac{\lambda}{R}$$

where $\lambda \in (0,0.85)$ yields positive phase margin. When λ decreases margins increase.

In order that the open-loop transfer function satisfies the crossover condition $\left|L(j\omega_g) = 1\right|$, K_{PI} is set to

$$K_{PI} = \omega_g z \left| \frac{j\omega_g + \frac{1}{R}}{\frac{C^2}{2N}} \right|$$

The s domain transfer function (3.6) needs to be converted into a z-domain transfer function for a digital implementation. A PI transfer function of the form (3.6) yields a z-domain transfer function between δp and δq of the form [32]

$$\frac{p(z)}{\delta q(z)} = \frac{a * z - b}{z - 1}$$

where $\delta q = q - q_{ref}$ with the target queue size q_{ref}, and $\delta p = p$ assuming target drop probability $p_{ref} = 0$.

Then the PI controller can calculate the packet drop probability at every sampling interval as follows.

$$p(t+1) = p(t) + a * (q(t+1) - q_{ref}) - b * (q(t) - q_{ref}) \tag{3.7}$$

3.4.3 Discussions

A PI controller calculates the packet drop probability directly and thus provides faster responses than RED. It also removes steady-state errors and demonstrates fair robust-

ness in a wide range of changed network conditions such as traffic load, link capacity and round-trip time. It can use a fixed sampling frequency lower than that of RED by 2-3 orders of magnitude. Thus the controller can reduce computations over RED significantly.

On the other hand, removing the threshold of RED may results in a higher loss rate. This is why the PI controller performs better with ECN than without ECN. It seems that the tuning problem of its coefficients *a, b*, and sampling interval corresponding to changed scenarios is not as severe as that of RED. However, the choice of its parameters is still the result of a trade-off between fast response and stability of the system under extensively varying network conditions.

3.5 Random Exponential Marking

3.5.1 Objectives

RED measures congestion by average queue size. So its performance measure like delay is coupled with the congestion measure. Since the average queue size reflects the change in traffic load, it is difficult for RED to maintain performance within the target zone regardless of the number of users. By using congestion measure independent of the performance measure, REM [5] aims to achieve high throughput, low loss rate and low queuing delay under varying traffic loads.

3.5.2 Congestion measure

Congestion measure indicates excess demand for bandwidth. The key of REM is to use price as congestion measure to determine the marking probability. Based on rate mismatch (i.e., difference between input rate and link capacity) and queue mismatch (i.e., difference between queue size and target), price reflects not only current congestion but also the change in congestion. At a queue l, the price $price_l(t)$ is updated in every period t [5] as follows.

$$price_l(t+1) = \left[price_l(t) + \gamma(\alpha_l(q_l(t) - q_{ref,l}) + x_l(t) - C_l(t))\right]^+ \quad (3.8)$$

where $x_l(t)$ is the aggregate input rate at queue l, $C_l(t)$ is the available bandwidth at queue l, and $[z]^+ = \max\{z,0\}$. The constant α_l is a small positive number. It can be set individually at each queue to trade off utilization and queuing delay during transient period. The constant γ is a small positive number. It controls the responsiveness of REM to changes in network scenarios.

If the target queue size $q_{ref,l}$ is nonzero and there are packets in the buffer, $x_l(t) - c_l(t)$ is the rate at which the queue size grows. In this case, it can be substituted by $q_l(t+1) - q_l(t)$. Thus the price is updated based only on the current and previous queue sizes [5] as follows.

$$price_l(t+1) = \left[price_l(t) + \gamma(q_l(t+1) - (1-\alpha_l)q_l(t) - \alpha_l q_{ref,l})\right]^+ \quad (3.9)$$

3.5.3 Sum prices

REM uses the following exponential marking policy to calculate the marking probability $p_l(t)$ [5].

$$p_l(t) = 1 - \phi^{-price_l(t)} \quad (3.10)$$

where $\phi > 1$ is a constant.

If an incoming packet is not marked at an upstream queue, its marking probability is exponential increasing in the current price. Assume that the link prices are small and hence the link marking probabilities are small. The end-to-end marking probability is calculated by the sum of link prices along the path [5] as follows.

$$1 - \prod_{l=1}^{L}(1 - p_l(t)) = 1 - \phi^{-\sum_l price_l(t)} \approx (\log_e \phi \sum_l price_l(t)) \quad (3.11)$$

Thus the end-to-end marking probability is an exponential function of the sum of individual congestion along the path. It can be observed by a source from marked packets and then the source can adapt its sending rate in response to the congestion.

In addition, other marking policies instead of exponential marking policy can be used to provide a modularized feature for REM.

3.5.4 Discussions

REM uses a price based on queue mismatch and rate mismatch as its congestion measure so that it is equivalent to the function of a PI controller. The measure of rate mismatch is like a proportional scheme, and the measure of queue mismatch is like an integral scheme that can remove static-state errors independent of network scenarios.

Therefore, REM can stabilize the queue at a target queue size in a wide range of varying network scenarios.

REM provides an exponential marking policy that can make use of the sum of prices along a path to calculate the end-to-end marking probability. However, this makes its parameters tightly coupled each other. On the other hand, although the authors in [5] thought that it could provide negligible queuing delay, small target queue size would cause high loss rate. This is because the oscillations of the queue size would become severe as round-trip time or link capacity increase.

REM is designed for the ECN environment. It does not work well if networks are dominated by dropping packets. There are four parameters $\phi, \alpha, \gamma, q_{ref}$ that need to be set in REM. The robustness of these parameters under various network conditions is a research topic in the future.

3.6 BLUE

3.6.1 Objectives

It is important to avoid high loss rate and keep low queuing delay in the Internet. RED needs constant tuning of its parameters to achieve acceptable performance under varying network conditions. This work has proven to be difficult [13][12][48][57]. BLUE aims to reduce packet loss rate and buffer size requirement in a simpler way [19].

3.6.2 Algorithm

The key idea behind BLUE is to perform queue management directly based on packet loss and link utilization rather than on the instantaneous or average queue size.

BLUE directly uses packet loss and link utilization as congestion measure to set marking probability. In principle, BLUE decreases its marking probability in response to an empty queue or a link idle event, and increases its marking probability if the buffer overflows or if the queue size exceeds a certain threshold. Thus BLUE can effectively “learn” the correct marking probability. The use of the threshold provides room to accommodate transient bursts. The algorithm of BLUE [19] is described as follows.

Upon packet loss (or $q > threshold$) event:

if ($(now\_time - last\_update) > freeze\_time$)

$p = \min(1, p + \delta_1)$

$last\_update = now\_time$

Upon link idle event:

if ($(now\_time - last\_update) > freeze\_time$)

$p = \max(0, p - \delta_2)$

$last\_update = now\_time$

Figure 3.3 BLUE algorithm

where δ_1 and δ_2 are increment and decrement respectively, *freeze_time* is the minimum time interval between two successive updates of the marking probability *p*, and *threshold* specifies the queue size above which the marking probability should be updated.

The use of *freeze_time* allows the changes in the marking probability to take effect before the value is updated again. It should be set based on the effective round-trip times of connections sharing the link in order to allow any changes in the marking probability to reflect back on to the end sources before additional changes are made, and should be randomized to avoid global synchronization.

δ_1 needs to be set much greater than δ_2. This is because link idle occurs when congestion management is either too conservative or too aggressive, but packet loss occurs only when congestion management is too conservative. By weighting heavily against packet loss, BLUE can quickly respond to a significant increase in traffic load.

On the other hand, δ_1 and δ_2 should be set in conjunction with *freeze_time* to allow p to range from 0 to 1 in an order of minutes for links where extremely large changes in load occur only in an order of minutes.

3.6.3 Discussions

BLUE can use a small buffer size to keep lower loss rate than RED. However, BLUE has a relatively sluggish response because there are many steps needed in correction of p to get its correct value and to stabilize the queue. In addition, BLUE has difficulty in stabilizing the queue under various round-trip times and link capacity. And research on parameter settings such as threshold is also needed.

3.7 Adaptive Virtual Queue

3.7.1 Objectives

Congestion results from the mismatch between input rate and outgoing rate at a link. This motivates AVQ [37] to control the input rate to be lower than outgoing rate by maintaining a desired link utilization. AVQ aims to achieve high link utilization with low delay and low loss rate at congested links.

3.7.2 Algorithm

A router maintains a virtual queue with capacity lower than the actual link capacity C. The buffer size of the virtual queue is equal to the real buffer size. Upon each packet arrival, the packet is enqueued in the real buffer and a copy of the packet is enqueued in the virtual buffer if there is sufficient room in the virtual buffer. If the fictitious packet overflows the virtual queue, the fictitious packet is dropped in the virtual queue and the real packet is marked or dropped in the real buffer, depending upon the congestion notification mechanism used by the router.

At each packet arrival event, the virtual queue capacity $\tilde{C}$ is updated according to the following differential equation [37]:

$$\dot{\tilde{C}} = \varphi(\tau C - x) \tag{3.12}$$

where x is the aggregate input rate at the link, $\tau \leq 1$ is the desired link utilization and $\varphi > 0$ is the smoothing parameter. The rationale behind this rate adaptive equation is that marking needs to become more aggressive when the actual link utilization exceeds the desired utilization and needs to become less aggressive when the actual link utilization is below the desired utilization. The detailed AVQ algorithm [37] is described as follows.

At each packet arrival epoch do

$VQ \leftarrow \max(VQ - \widetilde{C}(t-s),0)$ /*Update Virtual Queue Size */

If $VQ + b > B$

Mark of drop packet in the real queue

else

$VQ \leftarrow VQ + b$ /*Update Virtual Queue Size*/

endif

$\widetilde{C} = \max(\min(\widetilde{C} + \varphi * \tau * C(t-s), C) - \varphi * b, 0)$ /*Update Virtual Capacity */

$s \leftarrow t$ /*Update last packet arrival time*/

B: buffer size;

b: packet size in bytes;

VQ: virtual queue size in bytes.

s: arrival time of previous packet;

t: current time;

Figure 3.4 AVQ algorithm

3.7.3 Discussions

An AVQ algorithm is a rate-based mechanism different from RED which detects congestion based on the queue size. Simulations demonstrate that it could provide low queuing delay, low loss rate and high link utilization in the presence of long-lived flows and of short-lived flows.

However, there are two parameters that have to be chosen to implement AVQ: the desired utilization and the smoothing factor. The desired utilization allows an ISP to provide a trade-off between high levels of utilization and small queue lengths, and the smoothing factor determines how fast to adapt the marking probability at the link to the changing network conditions. Both parameters determine the stability of the AVQ queue. How to tune these two parameters in changing network scenarios is a topic for future research.

On the other hand, it is a drop-tail queue in essence. Hence it is liable to have the same problems that drop-tail induces. For example, it cannot avoid global synchronization when many packets are discarded in the virtual queue and then are marked in the real queue. It need further research to demonstrate whether or not the AVQ could avoid bias against bursty traffic and other fairness problems.

3.8 Conclusions

ARED, Auto-Tuning RED, PI, REM, BLUE and AVQ can provide better performance than RED. One reason is that their parameters are more robust than RED under varying network conditions, especially under varying traffic load. However, as the bandwidth-delay product increases, neither of them can keep the system stable, nor provide good transient performance. So the following chapters aim to design parameter tuning algorithms for the prominent AQMs in order to keep stable queue and desired transient performance under highly dynamic networks.

CHAPTER 4

The Mechanism of Adapting RED Parameters to TCP Traffic

4.1 Introduction

A critical unsolved problem of RED described in Chapter 2 is that its average queue size varies with traffic load as well as round-trip time and link capacity, and parameterizing RED to obtain good performance under variable congestion scenarios is very difficult. Such difficulties discourage network administrators from activating RED in their routers [48]. Certainly, there are many parameter tuning techniques for RED proposed in the literature [22][31][52][54][61][63]-[66], but they were either developed on the basis of empirical investigations and analysis, or are only applicable under certain assumptions. In particular, the authors in [22][52][54][61] provided guidelines for adjusting only one of the RED parameters for the changing network conditions. So these guidelines are applicable only under a narrow range of round-trip times and link capacities. Even so, the Web performance of other AQMs is not as good as ARED when dropped packets are used as indications of congestion [41], and their transient response would become much slower when network scenarios change dynamically [57].

The authors in [31] proposed a method to set initial RED parameters from a Control Theoretic perspective. However, they did not illustrate how to adapt these RED parameters to changing network scenarios. RED parameters are still coupled to each

other. By improving their Control Theoretic analysis of TCP/RED systems, this chapter developed an Auto-Parameterization RED (AP-RED) [12] to provide a simple, scalable and systematic algorithm for tuning these RED parameters as a function of network traffic conditions of link capacity, round-trip time, and the number of TCP flows. Theoretic analysis and nonlinear simulations using a ns-2 simulator [67] have demonstrated that it is robust, adaptive to TCP dynamics, and produces desirable transient performance.

The rest of this Chapter is organized as follows. Section 4.2 illustrates a predeveloped nonlinear dynamic model of TCP and a linear feedback model of TCP/RED system. Section 4.3 describes AP-RED and presents stability analysis. Section 4.4 introduces the calculation method for network parameters. Simulation results to validate the algorithm are presented in Section 4.5. Conclusions are made in Section 4.6.

4.2 Model

In [46], a nonlinear dynamic model of TCP behaviour was developed using fluid-flow and stochastic differential equation analysis. By ignoring timeout mechanism, a simplified version is used in [31] to describe the model by the following differential equations:

$$\begin{aligned} \dot{W}(t) &= \frac{1}{R(t)} - \frac{W(t)W(t-R(t))}{2R(t-R(t))}\, p(t-R(t)) \\ \dot{q}(t) &= \frac{W(t)}{R(t)} N(t) - C \end{aligned} \tag{4.1}$$

where $\dot{x}$ denotes the time-derivative of x and

W = expected TCP window size (packets);

q = expected queue size (packets);

R = round-trip time (seconds);

C = link capacity (packets/second);

N = number of TCP flows;

p = probability of packet mark/drop;

According to these equations, [31] further describes the behaviour of $\delta W = W - W_0$, $\delta q = q - q_0$, and $\delta p = p - p_0$ in a linear feedback control model of TCP/RED by linearizing variables (W, q, p) at its equilibrium point (W_0, q_0, p_0). The linear control model is depicted in Figure 4.1.

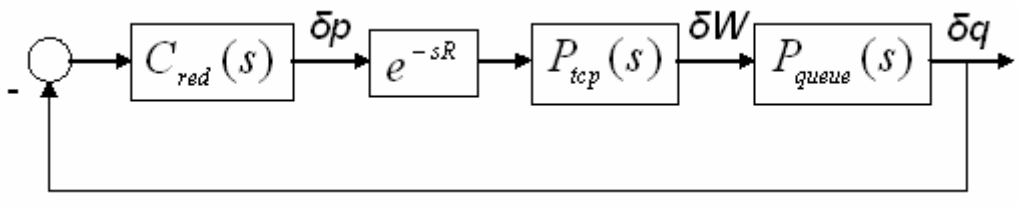


Figure 4.1 Feedback control model of TCP/RED system

In the above model $P_{tcp}(s)$ denotes the linearized TCP dynamics, $P_{queue}(s)$ denotes the queue dynamics, e^{-sR} denotes the delay term and $C_{red}(s)$ denotes RED control strategy. They were given by the following equations:

$$\begin{cases} P_{tcp}(s) = \dfrac{\dfrac{RC^2}{2N^2}}{s + \dfrac{2N}{R^2 C}} \\ P_{queue}(s) = \dfrac{\dfrac{N}{R}}{s + \dfrac{1}{R}} \\ C_{red}(s) = \dfrac{L_{red}}{s/K + 1} \end{cases} \tag{4.2}$$

where

$$L_{red} = \frac{p_{\max}}{\max_{th} - \min_{th}};$$

$K = -\frac{\log_e(1-w_q)}{T_s}$;

$p_{\max}$ = maximum drop probability;

$\max_{th}$ = maximum threshold;

$\min_{th}$ = minimum threshold;

w_q = average queue weight;

T_s = sampling interval.

4.3 AP-RED Algorithm and Stability Analysis

4.3.1 The algorithm

The objective of AP-RED is to stabilize the instantaneous queue size by maintaining the average queue size within the limits of $\max_{th}$ and $\min_{th}$, and to provide high link utilization in widely varying network scenarios. The rationale behind setting $\max_{th}$ and $\min_{th}$ as a fraction of bandwidth-delay product is to follow the theoretic analysis and rule-of-thumb in accommodating bursty traffic [22][26][52]. The rational behind setting $p_{\max}$ is to achieve a desirable equilibrium point based on theoretic analysis. The rationale behind setting w_q is to maintain the stability of the system from a Control Theoretic perspective.

Thus, consider the initial RED parameters w_{q0}, $p_{\max 0}$, $\max_{th0}$, and $\min_{th0}$ that stabilize the queue under the initial network scenario of TCP load N_0, round-trip time R_0 and link capacity C_0. When the network scenario varies we present the following algorithm for adjusting RED parameters.

$$\max_{th} = k_r k_c \max_{th0} \tag{4.3}$$

$$\min_{th} = k_r k_c \min_{th0} \tag{4.4}$$

$$p_{\max} = (\frac{k_n}{k_r k_c})^2 p_{\max 0} \tag{4.5}$$

$$w_q = \begin{cases} \dfrac{k_n}{(k_r k_c)^2} w_{q0} & N \le RC/2 \\ \dfrac{1}{k_r k_c} w_{q0} & N > RC/2 \end{cases} \tag{4.6}$$

where

$k_r = R / R_0$;

$k_c = C / C_0$;

$k_n = N / N_0$

and constraint $p_{\max}$ within the range [0.01,0.5]

i.e. if $p_{\max} > 0.5$ $p_{\max} = 0.5$

if $p_{\max} < 0.01$ $p_{\max} = 0.01$

Compared with the automatic setting of average queue weight w_q based on link capacity in [22], our tuning algorithm of w_q is based not only on link capacity, but also on round-trip time and traffic load. As w_q is adapted to congestion scenarios, a larger w_q can improve transient response while a smaller w_q provides a sufficient stability margin [31]. When $\max_{th}$ and $\min_{th}$ are reduced according to the changing network parameters, the queuing delay is decreased correspondingly. In contrast, the average queue size can still be stabilized within a given target range by using a larger $\max_{th}$ and $\min_{th}$ when round-trip time increases or link capacity increases.

4.3.2 Determining the equilibrium point

The purpose of adjusting $p_{\max}$ is to keep the queue at a desirable equilibrium point. At a new equilibrium point where $\dot{W} = 0$ and $\dot{q} = 0$, from (4.1) we have

$$W^2 p = 2 \text{ and } W = \frac{RC}{N}$$

Then we obtain

$$p = \frac{q - \min_{th}}{\max_{th} - \min_{th}} p_{\max} = 2(\frac{N}{RC})^2 \tag{4.7}$$

Hence at the equilibrium point the drop probability is determined by network parameters rather than other RED parameters.

Similarly, at the initial equilibrium point q_0 we obtain

$$p_0 = \frac{q_0 - \min_{th0}}{\max_{th0} - \min_{th0}} p_{\max 0} = 2(\frac{N_0}{R_0 C_0})^2 \tag{4.8}$$

Since we tune $p_{\max}$ in terms of (4.5), from (4.7) and (4.8) we have

$$q = k_r k_c q_0$$

Thus the equilibrium point q moves in proportion to the $\max_{th}$ and $\min_{th}$. It still stays between $\max_{th}$ and $\min_{th}$ when network condition changes.

4.3.3 Stability analysis

- Stability proposition

In this section we derive a simplified version of stability proposition based on the analysis given in [31].

Stability Proposition: Let L_{red} and w_q satisfy:

$$L_{red} w_q \le \begin{cases} \dfrac{0.8N^3}{(RC)^5}, & N \le RC/2 \\ \dfrac{0.4N^2}{(RC)^4}, & N > RC/2 \end{cases} \tag{4.9}$$

where

$$w_q << 1$$

Then, the linear feedback control system shown in Figure 4.1 is stable.

Proof:

Consider the frequency response of the compensated loop transfer function

$$L(j\omega) = C_{red}(j\omega)P_{tcp}(j\omega)P_{queue}(j\omega)e^{-j\omega R}$$

$$= \frac{L_{red}\dfrac{(RC)^3}{(2N)^2}e^{-j\omega R}}{(\dfrac{j\omega}{K}+1)(\dfrac{j\omega}{\dfrac{2N}{R^2C}}+1)(\dfrac{j\omega}{\dfrac{1}{R}}+1)}$$

$$\approx \frac{L_{red}\dfrac{(RC)^3}{(2N)^2}}{\dfrac{j\omega}{K}+1}e^{-j\omega R} \quad \forall \omega \in [0, \omega_g] \tag{4.10}$$

where

$$\omega_g = 0.1\min\left\{\frac{2N}{R^2C}, \frac{1}{R}\right\} \tag{4.11}$$

Thus we obtain

$$\left|L(j\omega_g)\right| \approx \frac{L_{red}\dfrac{(RC)^3}{(2N)^2}}{\sqrt{\dfrac{\omega_g^2}{K^2}+1}} < \frac{L_{red}\dfrac{(RC)^3}{(2N)^2}}{\dfrac{\omega_g}{K}} \le 1 \tag{4.12}$$

if

$$\frac{L_{red}(RC)^3}{(2N)^2} \le \frac{\omega_g}{K} \tag{4.13}$$

We again use (4.10) to obtain

$$\angle L(j\omega_g) \approx \angle \frac{L_{red}\dfrac{(RC)^3}{(2N)^2}}{\dfrac{j\omega_g}{K}+1} - \omega_g R \ge -90^0 - 0.1\frac{180^0}{\pi} > -180^0$$

This and (4.12) indicate that the closed-loop system is asymptotically stable according to the Nyquist stability criterion [30].

Given $w_q << 1$, we have

$$\log_e(1-w_q) \approx -w_q \tag{4.14}$$

For a stable congested queue we have

$$T_s = \frac{1}{C} \tag{4.15}$$

Thus, from (4.2), (4.14) and (4.15) we have

$$K = w_q C \tag{4.16}$$

Consequently, given any $N \le RC/2$, from (4.13) we obtain

$$\frac{L_{red}(RC)^3}{(2N)^2} \le \frac{\dfrac{0.2N}{R^2C}}{w_q C}$$

i.e.,

$$L_{red} w_q \le \frac{0.8N^3}{(RC)^5}$$

Similarly, given any $N > RC/2$, we obtain

$$L_{red} w_q \le \frac{0.4N^2}{(RC)^4}$$

This completes the proof.

- Stability analysis

For initial RED parameters w_{q0}, $p_{\max 0}$, $\max_{th0}$, $\min_{th0}$ that satisfy (4.9), given $N_0 > R_0 C_0 / 2$, we have

$$\frac{p_{\max 0}}{\max_{th0} - \min_{th0}} w_{q0} < \frac{0.4 N_0^2}{(R_0 C_0)^4} < \frac{0.8 N_0^3}{(R_0 C_0)^5}$$

and given $N_0 \le R_0 C_0 / 2$, we still have

$$\frac{p_{\max 0}}{\max_{th0} - \min_{th0}} w_{q0} < \frac{0.8 N_0^3}{(R_0 C_0)^5}$$

Given $N \le RC / 2$, from (4.3)-(4.6) we have

$$L_{red} w_q = \frac{p_{\max}}{\max_{th} - \min_{th}} w_q = \frac{k_n^3}{(k_r k_c)^5} \frac{p_{\max 0}}{\max_{th0} - \min_{th0}} w_{q0}$$

$$< \frac{k_n^3}{(k_r k_c)^5} \frac{0.8 N_0^3}{(R_0 C_0)^5} = \frac{0.8 N^3}{(RC)^5}$$

Similarly, Given $N > RC / 2$ we have

$$L_{red} w_q < \frac{0.4 N^2}{(RC)^4}$$

So the stability proposition in (4.9) can always be satisfied when we tune RED parameters according to equations (4.3)-(4.6). This provides a useful guideline for tuning RED parameters whilst maintaining the stability.

The derivation of our stability proposition is similar to that proposed in [31]. So we have an intuitive sense that our algorithm can provide a similar performance to that in [31]. However, the RED parameters in [31] are tightly coupled to each other and a complex calculation is needed to obtain suitable RED parameters for every changed network scenario. In contrast RED parameter tuning becomes much easier with the algorithm as shown in (4.3)-(4.6).

4.4 Network Traffic Measurement

The online statistic characteristics of heterogeneous network scenarios used in AP-RED include round-trip time *R*, number of TCP flows *N* and link capacity *C*. Based

on packet-level traces and flow-level statistics we know they are measurable and do not change dramatically over a relatively short time. For example, the characteristics of R, N, C and average package size etc have been observed in OC3MON [55] and the IPMON system [29]. It is feasible for us to use the methods described in the literature to obtain the network parameters.

In [29] the number of flows is calculated per minute. Those packets with the same 5-tuple information (source address, source port, destination address, destination port, and protocol) are classified as the same flow. The start time of a flow is the first time when a packet with new 5-tuple information is observed, and the flow ends when no packets with the same 5-tuple information are seen for a time interval 60s. On the other hand we can use a shorter time interval to calculate the traffic loads. Although there is some difference for the calculated value of traffic loads with different time intervals, in terms of [35] the variation of proportionality factor k_n is small with different time intervals.

In [29][45] round-trip time is measured as the time elapsed between a synchronization (SYN) packet and the first ACK packet that completes the three-way handshake. This is a rough estimate because we only compute the round-trip times for flows of which we observe the SYN/ACK pair. Fortunately, the deviation of the estimate is minimized because we use a proportionality factor k_r. In addition, RED also has its own robustness under changed scenarios [44].

Moreover the link capacity C in bytes is a known value, so it is unnecessary to calculate it in most cases when AP-RED runs in byte mode.

The measurement methods can be deployed in routers. For example, Cisco NetFlow is configured on most Cisco routers to provide a highly condensed and detailed view of all real time network traffic [68]. Because the statistical characteristics of network parameters are not changing dramatically we do not need to predict the network scenario parameters R, N and C in advance and could measure these parameters less frequently. In practice we need to set a suitable interval time to obtain these net-

work parameters on a trial-and-error basis. Because it is easy to obtain these network parameters in edge routers, the feasibility of our RED tuning algorithm is ensured, especially in edge routers.

4.5 Simulations

Although AP-RED is based on the analysis of a linearized model, the algorithm is verified by using a *ns-2* simulator to capture the stochastic, nonlinear nature of the network dynamic. As the prominent algorithm for RED parameter tuning [40], ARED is compared with AP-RED in the simulations. These simulations use a simple dumbbell topology with a single bottleneck link as shown in Figure 4.2. The bottleneck link is shared by persistent FTP flows, short HTTP sessions and reverse-path traffic. Each HTTP sessions repeatedly make short file transfers. Between two consecutive transfers, there is a 'think time' that starts after the last byte of the first file has been acknowledged. The transfer size is exponentially distributed with a mean of twelve 1 KB-packets. The think time is exponentially distributed with a mean of 500ms. The presence of these short-lived flows introduces noise into the queue. The number of HTTP flows in our simulations is set to twice that of FTP flows. The reverse-path traffic consists of 20 persistent FTP flows. The presence of reverse-path traffic introduces ACK compression and the loss of ACK packets, and thus increases the burstiness of the forward-path traffic. In all simulations the average packet size is set to 500 bytes, and the buffer size is set to 500 packets which are large enough for the following network scenarios. AP-RED is verified by extensive simulations using different R, C and N, but only limited simulations can be presented in this section.

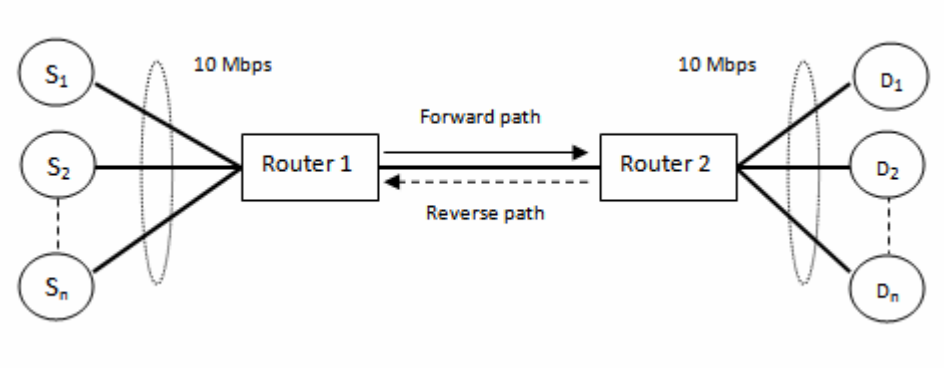
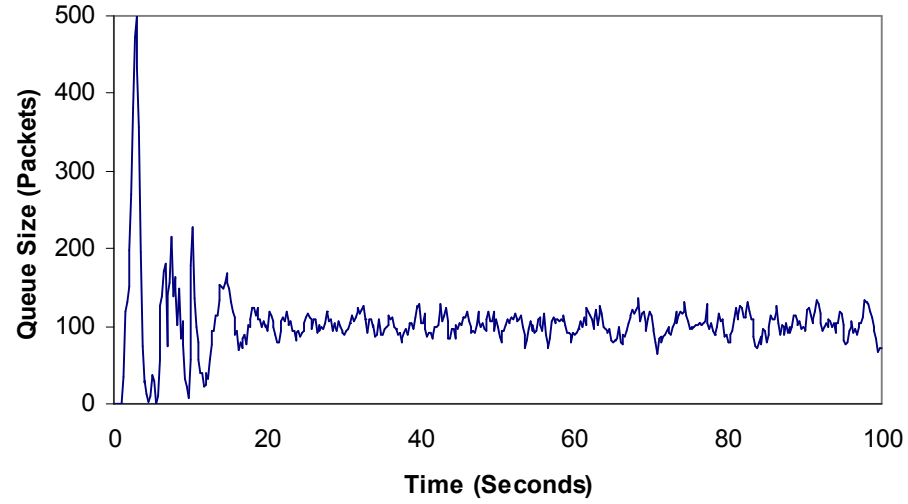


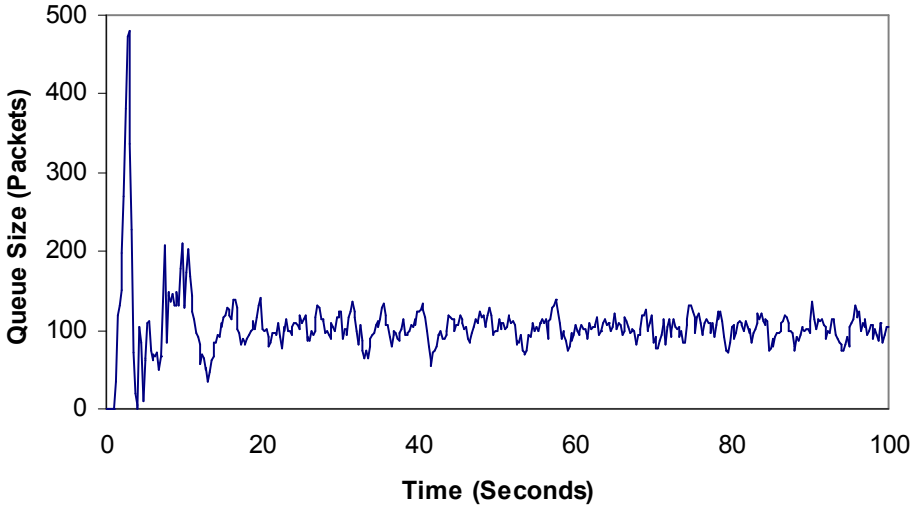
Figure 4.2 Network topology for simulations

4.5.1 Experiment 1

In the first experiment we have $N_0 = 50$ FTP flows with round-trip propagation delay $d_0 = 120$ ms. The bottleneck link capacity is set to 10 Mbps, that is, $C_0 = 2500$ packets/second. The corresponding parameters of AP-RED are $p_{\max 0} = 0.05$, $\min_{th0} = 50$ packets, $\max_{th0} = 150$ packets and $w_{q0} = 10^{-4}$, which satisfy the stability proposition (4.9). We choose $\min_{th}$, $\max_{th}$ and w_q for ARED the same as the corresponding parameters for AP-RED above, and keep these ARED parameters unchanged in the following experiments. Figure 4.3(a) plots the instantaneous queue size of ARED. Observe a stable queue with small fluctuations around the equilibrium point of 100 packets. Figure 4.3(b) shows that the instantaneous queue size of AP-RED is also stable. Since both ARED and AP-RED are essentially RED, we observe quite similar performance of their queues in Figure 4.3.



(a) ARED

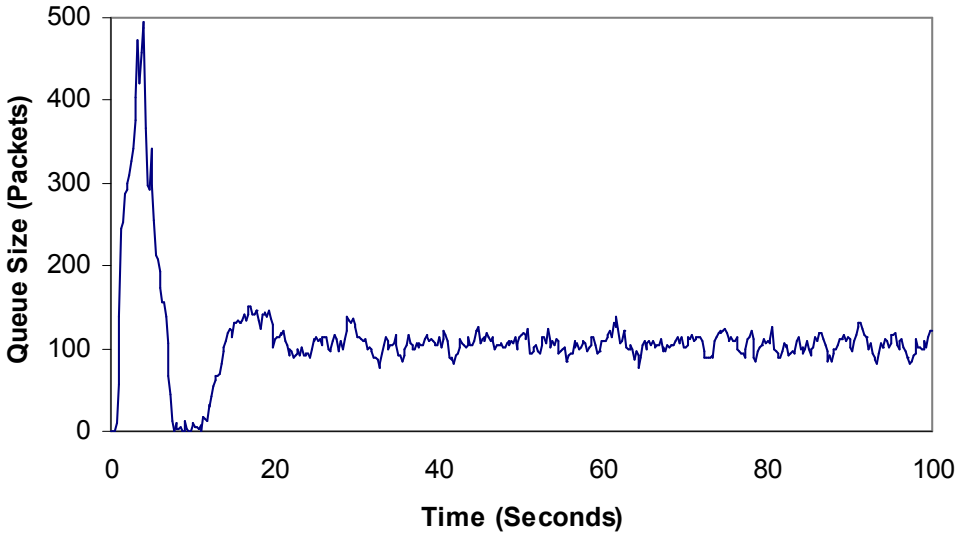


(b) AP-RED

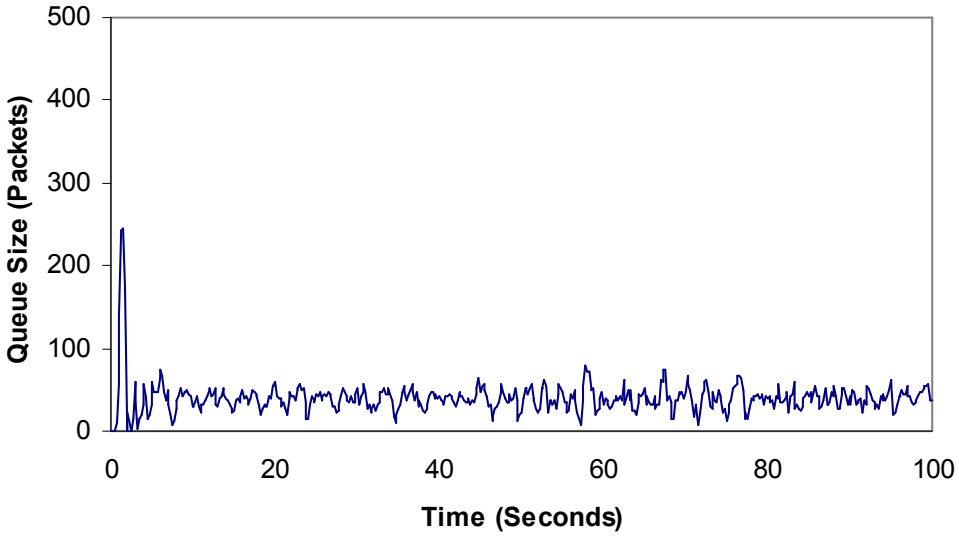
Figure 4.3 Comparison of ARED and AP-RED under network scenarios: *N*= 50, *C*= 2500 packets/second and *d* = 120ms.

4.5.2 Experiment 2

In the second experiment the round-trip propagation delay decreases to d = 100 ms and the link capacity decreases to C=1250 packets/second. The number of FTP flows decreases to N = 30. The plot in Figure 4.4(a) shows that the oscillations of the instantaneous queue size of ARED become smaller than that in Figure 4.3(a). This is because RED becomes more stable when round-trip time or link capacity decreases [44]. When we use d / d_0 to approximate k_r in AP-RED algorithm, $\min_{th} / C$ and $\max_{th} / C$ are in proportion to d according to equations (4.3) - (4.4). So are the queuing delay and the round-trip time. With the same network scenario, AP-RED parameters are changed to $p_{\max} = 0.1$, $\min_{th} = 20$ packets, $\max_{th} = 60$ packets and $w_q = 3.5 * 10^{-4}$, which are derived from the tuning algorithm (4.3)-(4.6). We can see that $\min_{th}$ and $\max_{th}$ decrease when R and C decrease. Figure 4.4(b) shows that the instantaneous queue size of AP-RED reduces to around 40 packets, compared to 100 packets in Figure 4.4(a), whilst the queue can be stabilized at the new equilibrium point between the updated $\min_{th}$ and $\max_{th}$. Consequently, a lower queuing delay is obtained. At the same time, the large average queue weight used in AP-RED should improve the transient response. As we observe, the queue settles to around its equilibrium point after 5 seconds, compared to 20 seconds in ARED.



(a) ARED

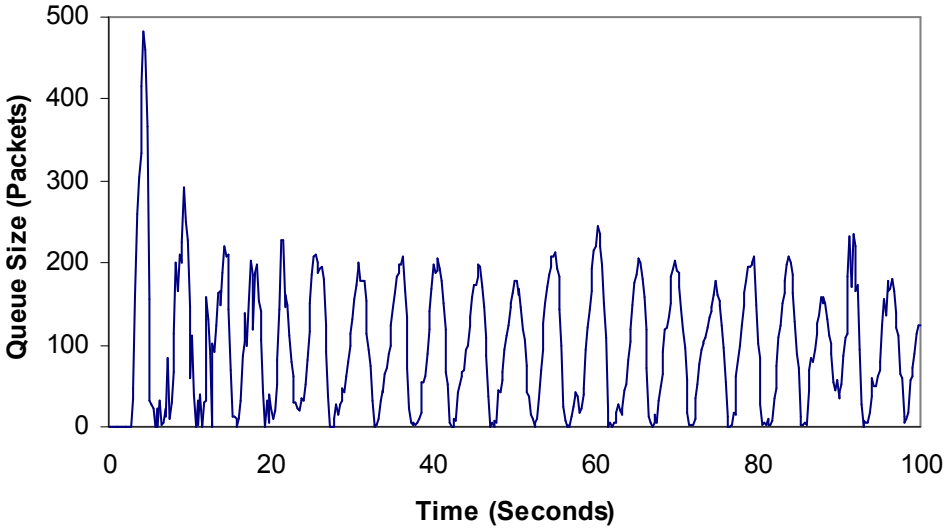


(b) AP-RED

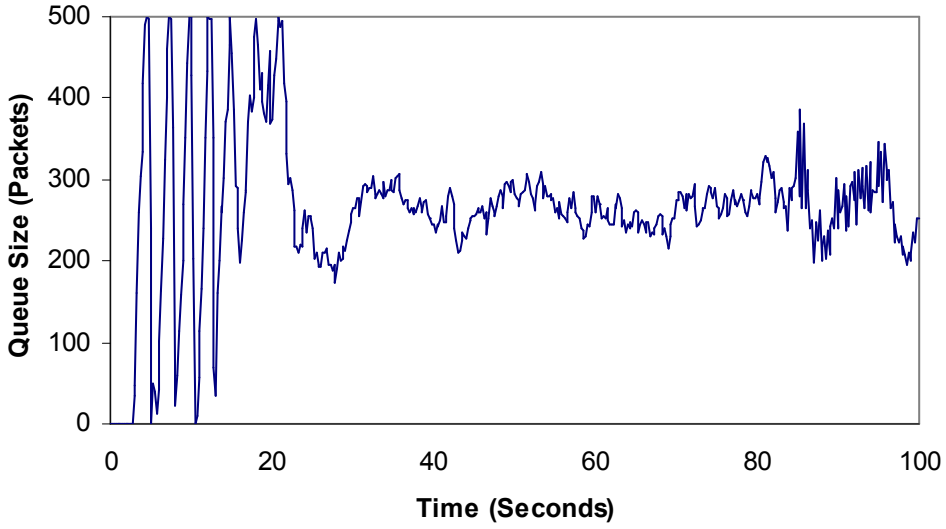
Figure 4.4 Comparison of ARED and AP-RED under network scenarios: N = 30, C = 1250 packets/second and d = 100ms.

4.5.3 Experiment 3

In the third experiment the round-trip propagation delay increases to d=250 ms and the link capacity increases to C=3000 packets/second. The number of FTP flows increases to N=100. The increase in the round-trip propagation delay and the link capacity should reduce the stability of the system or even make the system unstable [44]. The plot in Figure 4.5(a) displays deterministic oscillations of ARED, which lead to large delay jitter and low link utilization. With the same network scenario, AP-RED parameters are adjusted to $p_{\max} = 0.032$, $\min_{th} = 125$ packets, $\max_{th} = 375$ packets and $w_q = 3.2 * 10^{-5}$. These parameters are derived from the tuning algorithm (4.3)-(4.6) and still satisfy the stability proposition (4.9). We can see that $\min_{th}$ and $\max_{th}$ increase when R and C increase. Figure 4.5(b) displays that the instantaneous queue size of AP-RED can be stabilized around a new equilibrium point between the updated $\min_{th}$ and $\max_{th}$. Although the instantaneous queue size becomes large, it remains greater than zero and smaller than the total buffer size as desired. Thus high link utilization is maintained.



(a) ARED



(b) AP-RED

Figure 4.5 Comparison of ARED and AP-RED under network scenarios: *N*= 100, *C*= 3000 packets/second and *d* = 250ms.

4.6 Conclusions

In this Chapter we developed AP-RED to provide a scalable and systematic mechanism for RED parameter tuning. The advantages of AP-RED are that it provides a simple mechanism, from a theoretic standpoint, for constantly tuning four key RED parameters instead of tuning only one of them, like $p_{\max}$, under varying network conditions, such as traffic load, round-trip time and link capacity. It is unnecessary to consider the interaction among these RED parameters when they are changed in response to changed network conditions. Stability analysis and nonlinear simulations showed that AP-RED could stabilize the instantaneous queue size and maintain high link utilization in dynamic network scenarios. Multi-service situations involving networks with multiple bottlenecks and connections have not been considered in this study. More research is required in order to extend AP-RED to the multi-service and to a more realistic topology. The measure of network parameters may consume lots of CPU resources and memory resources in routers and make system complex. For this reason, Chapter 5 explores the feasibility of optimizing RED parameters under changing network scenarios without measurement of the network parameters *R*, *C* and *N*.

CHAPTER 5
Statistical Tuning RED in Dynamic Network Scenarios

5.1 Introduction

There are many RED parameter-tuning techniques proposed in the literature. A quantitative model for RED parameter-setting under various traffic load and round-trip times was developed in [63]-[66], but RED parameters are still tightly coupled with each other and need to be selected very carefully. A control-theoretic analysis has been developed in [31] to provide guidelines for tuning RED parameters under a specific network scenario, but it is difficult to adapt RED parameters to changing network scenarios. Adaptive RED (ARED) [22] and Auto-Tuning RED [52] provide a simple guideline for adjusting one of the key RED parameters in response to traffic load. However, the instantaneous queue size would become unstable when round-trip time or link capacity increases [10][44][59][63]-[66]. When bandwidth-delay product decreases the instantaneous queue size would be unnecessarily large [59][63]-[66]. The algorithm proposed in [59] explores to optimize the RED thresholds by providing tradeoffs with loss rate, but this iterative algorithm needs a long time to achieve the optimal values and thus becomes infeasible in highly dynamic networks. The authors in [60][63]-[66] try to tune several or even all key RED parameters, but these methods depend on the measurement of network parameters. This would increase overhead of CPU processing in routers or need additional software and hardware, and thus make

the system complex. In addition, the rough estimation of these network parameters [29][45][60] would increase uncertainty in tuning RED parameters.

Perhaps daunted by the difficulty of constantly tuning RED parameters to maintain good performance under dynamic networks, other AQM or AQM-based approaches have been provided. However, it is still difficult to design optimal parameters to achieve both very good steady-state performance and fast transient response in different network scenarios [57]. It has been demonstrated that ARED still provides the best performance when dropped packets are used as indications of congestion.

We think it is more important to solve the parameter tuning issue of existing prominent AQMs than to propose a new AQM algorithm. By investigating the stability analysis and the mechanism for parameter tuning described in Chapter 4, this chapter proposes that the change in the amplitude of queue oscillations mainly results from the change in round-trip time or link capacity. Thus we can measure the variance of the instantaneous queue size rather than measure network parameters to determine the adjustment factor of AP-RED in Chapter 4. Then a new method is designed to calculate the steady-state variance of the instantaneous queue size and to detect the change in steady-state variance. In this way, we propose Statistical Tuning RED (ST-RED) that can statistically adapt RED parameters in response to the detected change in the variance. Extensive simulations by using a ns-2 simulator demonstrate that it is robust and adaptive to TCP dynamics, and produces desirable transient performance.

The rest of this chapter is organized as follows. Section 5.2 investigates the background knowledge of TCP/RED system as well as an abruption detection technology. Section 5.3 describes the details of the ST-RED algorithm. Section 5.4 discusses the configuration of its initial parameters. We use ns-2 simulations to verify the algorithm in Section 5.5. Conclusions are presented in Section 5.6.

5.2 Background and Related work

The original RED algorithm [21] uses a low-pass filter with average queue weight w_q ($w_q << 1$) to calculate the average queue size $\tilde{q}$ at every packet arrival. When $\tilde{q}$ is less than a minimum threshold $\min_{th}$, no packets are marked or dropped. When $\tilde{q}$ is between $\min_{th}$ and a maximum threshold $\max_{th}$, arriving packets are marked or dropped with probability p, which varies linearly between 0 and a maximum mark/drop probability $p_{\max}$. When $\tilde{q}$ is greater than $\max_{th}$, all arriving packets are marked or dropped.

The stability proposition of TCP/RED system in Chapter 4 is given as follows:

Stability Proposition: Let L_{red} and w_q satisfy:

$$L_{red} w_q \leq \begin{cases} \dfrac{0.8N^3}{(RC)^5}, & N \leq RC/2 \\ \dfrac{0.4N^2}{(RC)^4}, & N > RC/2 \end{cases} \tag{4.9}$$

where

$$L_{red} = \frac{p_{\max}}{\max_{th} - \min_{th}} \text{ and } w_q << 1.$$

Then, the linear feedback control system shown in Figure 4.1 is stable. See proof in Chapter 4.

Obviously, four key RED parameters min_{th}, max_{th}, $p_{\max}$ and w_q, impact the stability of the system. In addition, the specific impact of these four parameters on the performance of the system is described in Chapter 2. The optimal values of min_{th} and max_{th} provide trade-offs between low queuing delay and high link utilization. $p_{\max}$

directly impacts upon the aggressiveness of RED and determines the position of the average queue size among min_{th} and max_{th}. w_q determines the degree of burstiness allowed in a router and determine the responsiveness of the system.

In Chapter 4, the mechanism for RED parameter tuning was developed using control-theoretical analysis. When initial round-trip time R_0 and initial link capacity C_0 vary to a new value of R and C, AP-RED can adjust $\max_{th}$ and $\min_{th}$ to new values according to the adjustment factor k as follows:

$$\begin{aligned} &\max_{th} \leftarrow k \max_{th} \\ &\min_{th} \leftarrow k \min_{th} \\ &\text{where } k = k_r k_c;\ k_r = \frac{R}{R_0};\ k_c = \frac{C}{C_0} \end{aligned} \tag{5.1}$$

If $N > RC/2$, AP-RED tune w_q as follows:

$$w_q \leftarrow \frac{1}{k} w_q \tag{5.2}$$

When $N \le RC/2$, the rule of thumb in [22][52] demonstrates that formula (5.2) can still be used to tune w_q.

In order to keep the average queue size within a target range $[\min_{th} + 0.4 * (\max_{th} - \min_{th}), \min_{th} + 0.6 * (\max_{th} - \min_{th})]$, ARED uses an AIMD policy to adjust $p_{\max}$ at every interval. Thus the instantaneous queue size can be stabilized around a reference value $q_{ref} = \frac{\min_{th} + \max_{th}}{2}$. In [22], the update interval is chosen as $5R$. In this way, the dynamics of ARED is still dominated by RED over a small time scale (on the arrival of every packet), and the adjustment of $p_{\max}$ is invoked over a large time scale. Detailed ARED algorithm see Figure 3.1.

In this chapter we use a Geometric Moving Average (GMA) decision function in [6] to detect significant changes in variance of instantaneous queue size. A decision func-

tion $\tilde{g}_k$ is defined in a recursive form to detect abrupt changes in the variance of variable y_k as follows:

$$\tilde{g}_k \leftarrow (1-\phi)\tilde{g}_{k-1} + \phi(y_k - \mu)^2 \tag{5.3}$$

where ϕ is a forgetting factor and μ is the mean value of y_k.

The alarm time t_a is defined by the following decision rule:

$$t_a = \min\{k : \tilde{g}_k \geq h\} \tag{5.4}$$

where h is a pre-defined threshold.

5.3 Algorithm

Because of the feedback nature of TCP's AIMD control strategy, oscillations in the queue size are very common [22]. Since the order of the bandwidth-delay product RC is much higher than that of the traffic load N in the stability proposition (4.9), round-trip time and link capacity make significantly more contribution than traffic load in the determination of the stability of TCP/RED system. On the other hand, analysis and experiments in [44] illustrate that the impact of other network conditions like "mice" traffic or reverse-path traffic on the queue oscillations is slight.

Consequently, we can infer that a significant change in the amplitude of the oscillations of the queue size mainly results from a significant change in the bandwidth-delay product. The main cause for larger oscillations of the queue size is the increase in the round-trip time or link capacity, while the main cause for smaller oscillations of the queue size is the decrease in the round-trip time or link capacity. As the variance is statistically related to the amplitude of the oscillations, this motivates ST- RED to use the variance of q, rather than R and C, to determine k used in equations (5.1) and (5.2). Thus the difficulty of measuring network variables can be avoided.

The overall guidelines for ST-RED are to stabilize the instantaneous queue size within a target range by maintaining steady-state variance of q greater than a lower threshold and less than an upper threshold. It is described as follows:

1) On every packet arrival, ST-RED uses a GMA function to calculate variance σ. At the same time, ST-RED calculates a GMA detection function with a lower threshold $h_s \cdot q_{ref}$ and an upper threshold $h_l \cdot q_{ref}$. The upper threshold is used to detect significantly large variance of q while the lower threshold is used to detect significantly small variance of q. The upper threshold is set to a steady-state value of the decision function. The variance function has the same input samples as the detection function so that the steady-state values of these two functions are similar. By setting the forgetting factor of the variance function greater than that of the detection function, the variance can increase faster than the detection function so that the variance already reaches its steady-state value (no less than the upper threshold) when the detection function hits the upper threshold. On the other hand, the variance can decrease faster than the detection function so that it reaches a value between the lower threshold and its steady-state value (no greater than the lower threshold) when the detection function hits the lower threshold. When large variance or small variance is detected RED parameters are adjusted according to (5.1)-(5.2) with k set to $\sigma/(h_l \cdot q_{ref})$ or $\sigma/(h_s \cdot q_{ref})$ correspondingly.

2) The variance is calculated when q is within l_range, a largest possible steady-state oscillation range of q. Calculations of the variance σ stop when q is out of this range, where q is treated as a transient-state. This is because the variance may become unnecessarily large during a transient period and thus may have not reached its steady-state value before the detection function reaches its upper threshold.

3) The detection function is calculated only when q is within d_range, a desirable steady-state oscillation range of q. This range is also used to prevent calculation of the transient-state value of q as much as possible. Otherwise these transient-state values

may result in the detection function operating incorrectly. Obviously d_range should be less than l_range.

4) At every interval, ST-RED uses the same AIMD policy of ARED to adjust $p_{\max}$.

The general algorithm for ST-RED is depicted in Figure 5.1.

Every packet arrival:

if $q < l\_range$

calculate the variance of q

if $q < d\_range$

calculate the variance detection function of q

if a significant change in the variance is detected

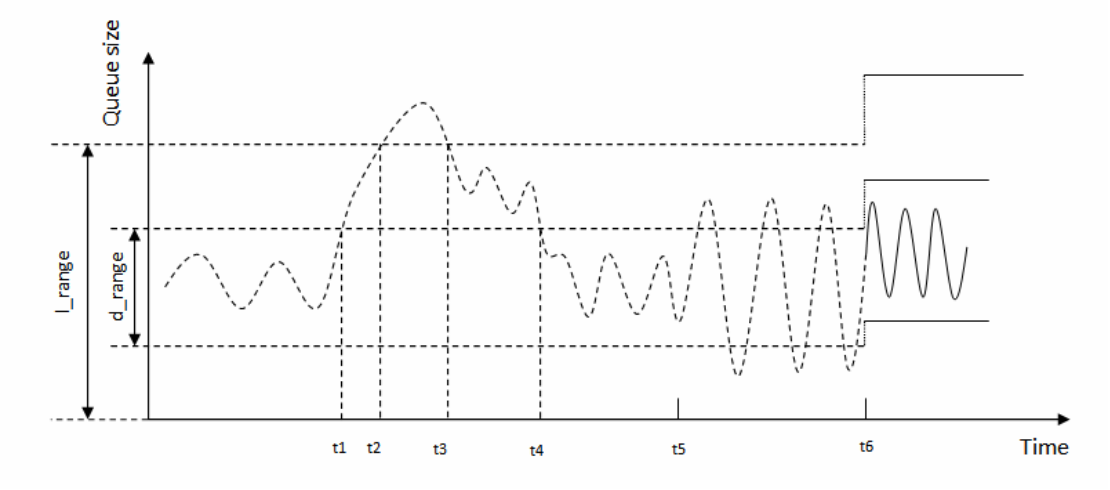
adjust $\max_{th}$, $\min_{th}$ and w_q according to the variance

Every *interval*:

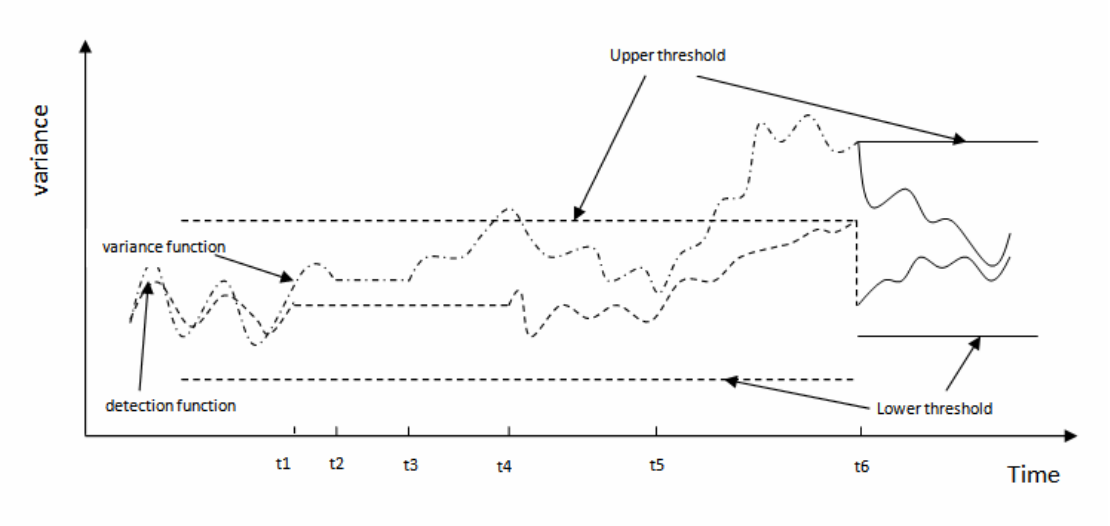
Use AIMD policy to adjust $p_{\max}$

Figure 5.1 General ST-RED Algorithm

Figure 5.2 illustrates an evolution of the queue size and the variance. At the beginning the queue size is within its target zone. So the detection function oscillates between the upper threshold and the lower threshold. We can see that the performance of the variance function is similar to that of the detection function. From t_1 to t_4 the queue size enters into a transient state. We see that the detection function stops calculation when the queue size exceeds d_range from t_1 to t_4 and the variance function stops calculation from t_2 to t_3 when the queue size is out of l_range. After the transient state ends at t_4, the oscillations of the queue size are benign. We see that the variance and the detection function oscillate within the thresholds. When large oscillations of the queue size occur after t_5, both the detection function and the variance increase.



a) Evolution of the queue size



b) Evolution of the variance function and detection function

Figure 5.2 Evolution of the variance and detection function

The variance increases faster than the detection function so that it reaches its steady state before the detection function hits the upper threshold. When the large variance is detected by the detection function at t_6, the upper threshold is moved to the current value of the variance by adjusting RED parameters $\max_{th}$ and $\min_{th}$. This results in an updated lower threshold. d_range and l_range are also updated corresponding to

the adjustment in $\max_{th}$ and $\min_{th}$. w_q is adjusted to a smaller value according to (5.2). Then the detection function is reset to a new initial value. From stability proposition (4.9) this adjustment makes the queue more stable. We see queue oscillations reduce after adjustment at t_6. Thus the detection function is adjusted to values less than the upper threshold and greater than the lower threshold as desired.

The detailed ST-RED algorithm is depicted in Figure 5.3 and explanation of the algorithm is followed. Unlike ARED, we do not restrict $p_{\max}$ to [0.01, 0.5]. Without the limitation, the transient period of ST-RED is still satisfied.

Initialization:

$\sigma^2 \leftarrow 0$ $\quad\quad$ $g^2 \leftarrow ((h_s \cdot q_{ref})^2 + (h_l \cdot q_{ref})^2)/2$

every packet arrival:

if $0 < q < 2 \cdot q_{ref}$

Calculate the variance:

$\sigma^2 \leftarrow (1 - \phi_1 \cdot w_q)\sigma^2 + \phi_1 \cdot w_q (q - q_{ref})^2$

if $(1-\gamma)q_{ref} < q < (1+\gamma)q_{ref}$

Calculate the detection function:

$g^2 \leftarrow (1 - \phi_2 \cdot w_q)g^2 + \phi_2 \cdot w_q (q - q_{ref})^2$

if ($g > h_l \cdot q_{ref}$ and $\sigma > h_l \cdot q_{ref}$) or

($g < h_s \cdot q_{ref}$ and $\sigma < h_s \cdot q_{ref}$)

Adjust parameters:

if $g > h_l \cdot q_{ref}$

$k \leftarrow \sigma / (h_l \cdot q_{ref})$

else

$k \leftarrow \sigma / (h_s \cdot q_{ref})$

endif

$\max_{th} \leftarrow k \cdot \max_{th}$ $\quad$ $\min_{th} \leftarrow k \cdot \min_{th}$

$w_q \leftarrow w_q / k$ $\quad\quad$ $interval \leftarrow k \cdot interval$

$g^2 \leftarrow ((h_s \cdot q_{ref})^2 + (h_l \cdot q_{ref})^2)/2$

endif

endif

endif

Every *interval*:

if ($\tilde{q}$ > *target*)

$p_{\max} \leftarrow p_{\max} + \alpha$

elseif ($\tilde{q}$ < *target*)

$p_{\max} \leftarrow p_{\max} * \beta$

endif

variables:

σ^2 :variance function of q g^2 : variance detection function of q

k: adjustment factor

Fixed parameters:

q_{ref} : $0.5 * (\max_{th} + \min_{th})$; target:

$[\min_{th} + 0.4 * (\max_{th} - \min_{th}), \min_{th} + 0.6 * (\max_{th} - \min_{th})]$;

α: min (0.01, $p_{\max}$ /4) ; β: 0.9;

ϕ_1 : 1 ; ϕ_2 : $0.25\phi_1$; γ : 0.7;

h_l : $\gamma / \sqrt{3}$; h_s : fixed in terms of initial RED parameters.

Figure 5.3 Detailed ST-RED algorithm

5.3.1 Calculating the variance and detection function of *q*

The following GMA function σ^2 can be used to calculate the variance:

$$\sigma^2 \leftarrow (1-\phi_1 \cdot w_q)\sigma^2 + \phi_1 \cdot w_q (q - q_{ref})^2 \tag{5.5}$$

where ϕ_1 is the forgetting factor for variance.

As the oscillations are symmetrically bounded around q_{ref} and the smallest possible steady-state value of q is 0, the largest possible steady-state value of q is $2 \cdot q_{ref}$. So the variance is calculated within l_range:

$$0 < q < 2 \cdot q_{ref} \tag{5.6}$$

According to (5.3), the following GMA detection function g^2 is used to detect significant changes in the variance of q:

$$g^2 \leftarrow (1-\phi_2 \cdot w_q) g^2 + \phi_2 \cdot w_q (q - q_{ref})^2 \tag{5.7}$$

where ϕ_2 is the forgetting factor for detection.

The detection function is calculated only when q is within d_range:

$$(1-\gamma) q_{ref} < q < (1+\gamma) q_{ref} \tag{5.8}$$

where γ is the coefficient of the detection range.

Since the amplitude of the queue oscillations needs to be maintained smaller than q_{ref} to provide high link utilization, we can choose the upper threshold as a fraction of q_{ref}. So we have the following decision rule according to (5.4) for large variance detection:

$$g > h_l \cdot q_{ref} \tag{5.9}$$

where $h_l \cdot q_{ref}$ is the upper threshold and h_l is the coefficient of the upper threshold.

Similarly, the decision rule for small variance detection is

$$g < h_s \cdot q_{ref} \tag{5.10}$$

where $h_s \cdot q_{ref}$ is the lower threshold and h_s is the coefficient of the lower threshold.

Although the initial value of the decision function g_0^2 is chosen as 0 in [6], it would be a value greater than $(h_s \cdot q_{ref})^2$ and less than $(h_l \cdot q_{ref})^2$ after a while if there are

not significant changes. For simple reason, we set the initial value to half way between $(h_s \cdot q_{ref})^2$ and $(h_l \cdot q_{ref})^2$. So we have

$$g_0^2 = \frac{(h_s \cdot q_{ref})^2 + (h_l \cdot q_{ref})^2}{2}.$$

5.3.2 Stability analysis

Consider that a large variance is detected and σ^2 reaches its steady state. We have

$$g \le \sigma \tag{5.11}$$

Then we adjust the upper threshold $h_l \cdot q_{ref}$ to σ by adjusting the RED parameters according to (5.1) and (5.2), where k is set to

$$k \leftarrow \frac{\sigma}{h_l \cdot q_{ref}} \tag{5.12}$$

From (5.9) and (5.11) we have $k > 1$.

Therefore after adjustment we have

$$g \le h_l \cdot q_{ref} \tag{5.13}$$

From the stability proposition (4.9) we know that the system becomes more stable for larger $\max_{th}$ and $\min_{th}$ and a smaller w_q. This indicates a smaller g and thus secures (5.13).

Similarly, when a small variance is detected we have

$$k \leftarrow \frac{\sigma}{h_s \cdot q_{ref}} \tag{5.14}$$

and after adjustment we have

$$g \ge h_s \cdot q_{ref} \tag{5.15}$$

The above analysis indicates that the queue size can be stabilized in a target range.

Remark: Because previous amplitude of queue oscillations are limited by q_{ref}, previous variance would increase greatly with the increase in previous q_{ref}. In this case, when RED parameters are adapted according to the adjustment factor (5.12) in re-

sponse to a large variance detected, further adjustment of RED parameters is needed until formula (5.13) is secured. On the other hand, when RED parameters are adapted according to the adjustment factor (5.14) in response to a small variance detected, variance σ may not reach its steady-state value. In this case, further adjustment of RED parameters is needed until σ reaches its steady-state value and formula (5.15) is secured.

We also take measures to prevent incorrect calculation of the adjustment factor *k* in exceptional cases. When large variance is detected, we need $\sigma > h_l \cdot q_{ref}$ to guarantee $k > 1$. When small variance is detected, we need $\sigma < h_s \cdot q_{ref}$ to guarantee $k < 1$.

5.4 Initial Parameter Setting

5.4.1 Choice of parameters *interval*, ϕ_1 and ϕ_2

Consider the change in link capacity is not significant for a specific link. We know that interval is determined by *R*. When RED parameters are adjusted according to (5.1) *interval* needs to be updated by:

$$interval \leftarrow k \cdot interval$$

The simulations and analysis in [22][52] indicate that w_q can be set to $\frac{1}{10RC}$. So from (4.16) we have $K << \frac{1}{R}$.

Then the delay term in the open-loop transfer function of TCP/RED (4.10) can be ignored. We have

$$L(j\omega) \approx \frac{L_{red}\dfrac{(RC)^3}{(2N)^2}}{\dfrac{j\omega}{K}+1}$$

Thus, we obtain the time constant of the closed-loop system in Figure 4.1

$$T \approx \frac{1/K}{L_{red}\frac{(RC)^3}{(2N)^2}+1} \le \frac{1}{K} = \frac{1}{w_q C} \tag{5.16}$$

The time constant of the estimator of the average queue size is referred to $\frac{-1}{\log_e(1-w_q)}$ packet arrivals in [22]. As the interval for packet arrival is $\frac{1}{C}$ for a stable queue, we can also define its time constant as $\frac{1}{w_q \cdot C}$ when $w_q << 1$. Similarly, the time constant of the variance function is $\frac{1}{\phi_1 \cdot w_q \cdot C}$ and the time constant of the detection function is $\frac{1}{\phi_2 \cdot w_q \cdot C}$. The time constant of the variance function should be greater than that of the system in order that steady-state values of the variance come from the steady-state values of the queue size. So from (5.5) and (5.16) we have $\phi_1 \le 1$. Our default setting of $\phi_1 = 1$ obeys these constraints. From (5.5) and (5.7) we have $\phi_1 > \phi_2$ to guarantee that the variance function increases or decreases faster than the detection function before a large variance or a small variance is detected, and that the variance has reached its steady state when the detection function settles to a final value of the upper threshold. On the other hand, ϕ_2 cannot be too small; otherwise, the detection function is too sluggish to detect significant changes quickly. Our default setting of $\phi_1 = 4\phi_2$ satisfies this constraint. This also indicates that although any smaller value of $\phi_1 < 1$ can be chosen, a smaller value of ϕ_2 has to be selected correspondingly. In this case it would take longer to detect the change in the variance of the queue size. So a significant change can be more quickly detected and the variance function can reach its steady-state value more quickly with the default setting of $\phi_1 = 1$.

5.4.2 Choice of parameters γ, h_l and h_s

Assume that the random variable q is a uniform variable. Its amplitude is $\sqrt{3}$ times its standard deviation. We choose h_l as $\gamma/\sqrt{3}$ in order to set the upper threshold in (5.9) to the standard deviation corresponding to the largest amplitude of $\gamma \cdot q_{ref}$ in d_range (5.8). Thus the upper threshold is a steady-state value that the detection function can reach.

A smaller q_{ref} in (5.9) can be used for a larger γ to keep the same g less than the upper threshold $q_{ref} \cdot \gamma/\sqrt{3}$, but the instantaneous queue size may begin to reach zero and downgrade link utilization for a smaller q_{ref}. On the other hand, a larger q_{ref} needs to be used to a smaller γ to satisfy (5.9). So the value of γ reflects the trade-off between high link utilization and low queuing delay. Based on extensive experiments, a default value of $\gamma = 0.7$ can be used to maintain high link utilization.

Furthermore, if h_s in (5.10) is too small it is difficult for the detection function to detect a small variance of q. In this case, low queuing delay as desired can't be provided in response to the small variance. If h_s is too large, the space between the upper threshold and the lower threshold is not large enough. In this case, when $\max_{th}$ and $\min_{th}$ increase in response to a large variance detected, the detection function tends to detect a small change, and vice versa. Thus q_{ref} may oscillate between different values.

h_s needs to be tuned mainly in terms of initial RED parameters. This is because the initial RED parameters determine the oscillations of q and will influence the subsequent oscillations corresponding to adapted RED parameters. In choosing h_s, we need to make sure that desirable low queuing delay can be provided in response to small variances of the queue size, and that the oscillations of q_{ref} between different

values should be avoided under stable network conditions. Once a suitable h_s is chosen, it remains unchanged under different network conditions.

5.5 Simulations

In this section, the performance of ST-RED is compared with prominent AQM mechanisms, which are ARED and PI controller, via *ns-2* simulations. Extensive simulations have been conducted, but only limited results can be reported in this section. These simulations use a simple dumbbell topology with a single bottleneck link as shown in Figure 4.2. The bottleneck link is shared by persistent FTP flows, short HTTP sessions, reverse-path traffic and UDP traffic. The presence of these short-lived flows introduces noise into the queue. The reverse-path traffic consists of persistent FTP flows. The presence of reverse-path traffic introduces ACK compression and the loss of ACK packets, and thus increases the burstiness of the forward-path traffic. There are 20 reverse-path flows in each simulation. In all simulations the byte mode is used and ECN is not supported. The initial parameters of ST-RED are designed for a scenario of 100 FTP flows passing through a 10Mbps bottleneck link. The round-trip propagation delays are uniformly 100ms. The packet size is 500 bytes. The corresponding parameters for ST-RED are $\min_{th} = 20$, $\max_{th} = 60$, $w_q = 5*10^{-5}$ and $h_s = 0.03/\sqrt{3}$. For ARED, we set w_q the same as that of ST-RED and vary $(\min_{th}, \max_{th})$ to test different queue performance. In all simulations the initial *interval* for ST-RED and *interval* for ARED are set to 0.5s. For PI controller, we use default values of a_=0.00001822 and b_=0.00001816 in [67]. Its sampling frequency is set to 170HZ. Different values of q_{ref} are used to test different queue performance. The performances of each scheme under changing network variables of round-trip time, link capacity, and traffic load are investigated. System performances under varying UDP traffic density and varying short-lived TCP flows are also studied. The buffer

size in all simulations is set to 500 packets which are sufficiently large for the following simulations.

5.5.1 Varying round-trip time

This experiment aims to investigate the performance of each scheme when the round-trip time varies. The round-trip time is varied by changing the propagation delay at the bottleneck link. At time 0 there are 100 FTP flows passing through a 10Mbps bottleneck link. The initial round-trip propagation delay d is 100ms. At the 200^{th} second, 400^{th} second, 600^{th} second and 800^{th} second, d increases to 300ms, 500ms, 700ms and 900ms respectively. At the 1000^{th} second, 1200^{th} second, 1400^{th} second and 1600^{th} second, d returns to 700ms, 500ms, 300ms and 100ms, respectively. Figure 5.4 plots the instantaneous queue size for each scheme.

For ARED (20, 60) with $\min_{th} = 20$ and $\max_{th} = 60$, the queue oscillations are benign from 0 second to 400 seconds. This demonstrates some robustness of ARED parameters under changing round-trip times. Between 400 seconds and 1400 seconds, the queue oscillations become severe, corresponding to d varying between 500ms and 900ms. For ARED (60, 180) deterministic oscillations can also be observed between 400 seconds and 1400 seconds. This is because that the interval of 0.5s is too small for ARED when the round-trip time becomes large.

For PI-40 with $q_{ref} = 40$, the figure shows its queue takes about 100 seconds to settle down. When d is less than 500 ms, the queue needs a long time to settle down in response to every change in d. So does PI-120. This is because the PI parameters which can stabilize the queue with large delay become too conservative when the delay is small. On the other hand, PI-120 can finally stabilize the queue with all d but PI-40 cannot do this when d is greater than 500 ms.

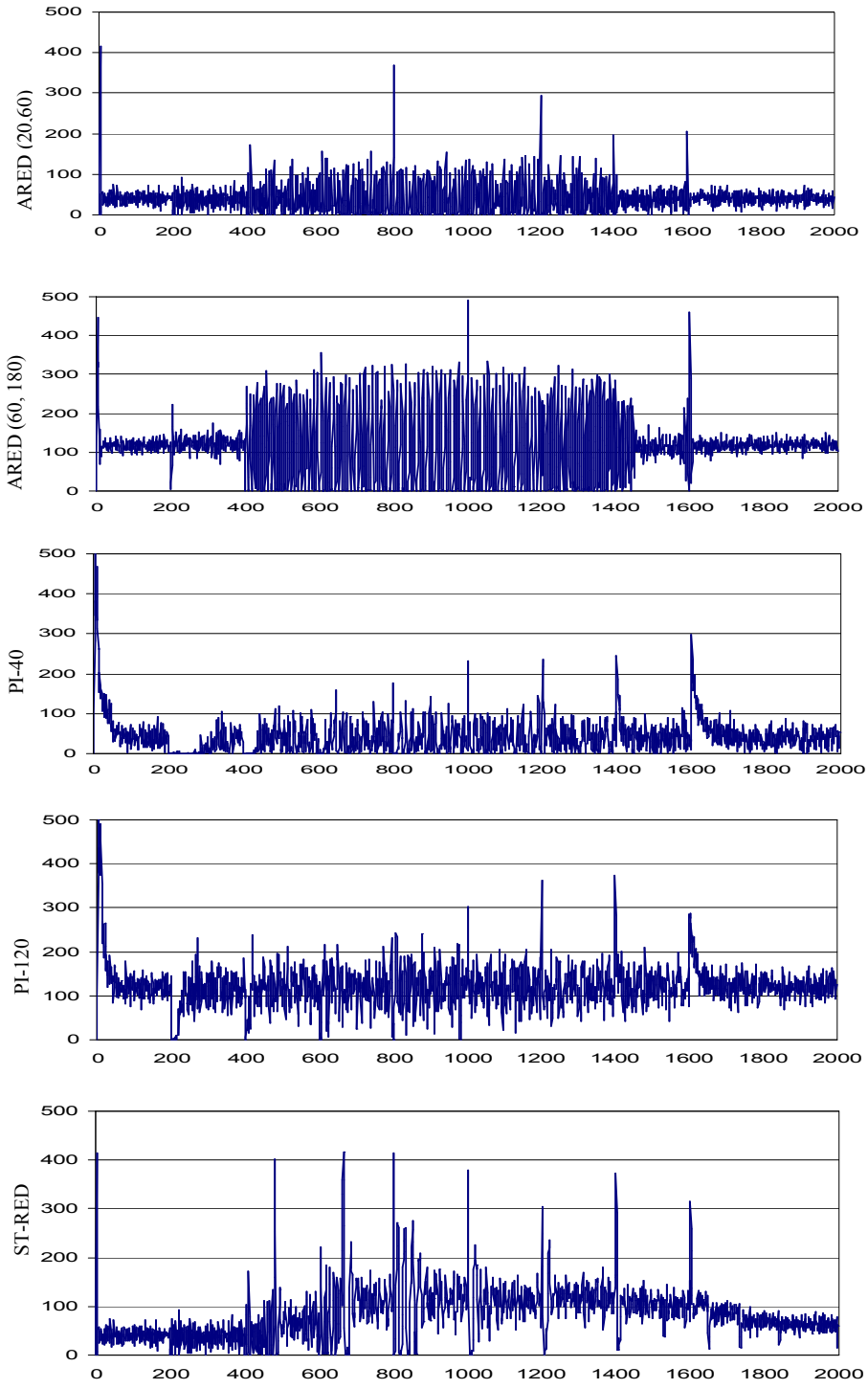


Figure 5.4 Queue size (packets) variations versus time (seconds) under varying round-trip propagation delay [100ms-900ms]

For ST-RED, the queue size increases to about 120 packets as *d* increases from 100ms to 900ms, and the queue size decreases as *d* decreases. It eventually returns to near 60 packets when *d* returns to its initial value 100ms. We can see a stable queue almost all the time.

Link utilization or throughput, average queue size, and average queue deviation are used as performance metrics. The average queue deviation is obtained by first calculating different values of queue deviation for every 200 seconds such as 1-200 seconds and 201-400 seconds, and then averaging these values. This is because q_{ref} for ST-RED may change significantly during the whole process, while it is relatively stable during a period of 200 seconds between two changes of network parameters.

Table 5-1 Summary Statistics for all Designs under Varying Round-trip Propagation Delay

	Link Utilization	Loss rate	Average queue size	Average queue deviation
ARED (20,60)	98.81%	3.83%	39.2	30.9
ARED (30,90)	98.75%	3.58%	59.4	41.9
ARED (40,120)	98,78%	3.38%	78.6	51.2
ARED (50,150)	98.94%	3.14%	98.9	58.5
ARED (60,180)	98.92%	2.99%	116.1	70.5
PI-40	97.77%	4.13%	42.5	37.8
PI-60	98.94%	3.80%	62.5	41.4
PI-80	99.37%	3.56%	82.2	44.2
PI-100	99.53%	3.33%	102.1	44.4
PI-120	99.71%	3.13%	121.6	44.3
ST-RED	99.67%	3.47%	83.9	40.9

In Table 5-1, we see that ST-RED provides very high utilization of 99.67%. Although ARED (50, 150), ARED (60, 180), PI-100 and PI-120 have lower loss rate than ST-RED, their average queue sizes are higher than ST-RED of 83.9 packets. On the other hand, the loss rate 3.47% of ST-RED is less than those of schemes with smaller average queue size, except ARED (40, 120). Although ARED (40, 120) has a slightly lower loss rate of 3.38% and a smaller average queue size of 78.6 packets, its link utilization of 98.78% and average queue deviation of 51.2 packets are much higher than those of ST-RED. As a result, ST-RED provides the best performances in this scenario.

5.5.2 Varying bottleneck link capacity

The objectives of the second experiment are to investigate the performance of each scheme under different bottleneck link capacities. The initial link capacity is 10Mbps. Then it increases to 20Mbps, 30Mbps, 40Mbps and 50Mbps at the 200th second, 400th second, 600th second and 800th second, respectively. From the 1000th second, it begins to decrease. It decreases to 40Mbps, 30Mbps, 20Mbps and 10Mbps at the 1000th second, 1200th second, 1400th second and 1600th second respectively. We set the number of FTP flows to 100 and the round-trip propagation delay to 100ms. The results for each scheme are given in Figure 5.5.

The plot of ARED (20, 60) shows a stable queue from 0 second to 200 seconds, and acceptable queue oscillations from 200 seconds to 400 seconds, This is because of the robustness of RED parameters. The figure shows malignant oscillations between 400 seconds and 1400 seconds when link capacity is between 30Mbps and 50Mbps. As the link capacity decreases to values no greater than 20Mbps after 1400 seconds, the queue size becomes stable again. For ARED (60, 180), although the queue oscillations become large as the link capacity increases, it is acceptable throughout the simulation. So it could provide high link utilization all the time.

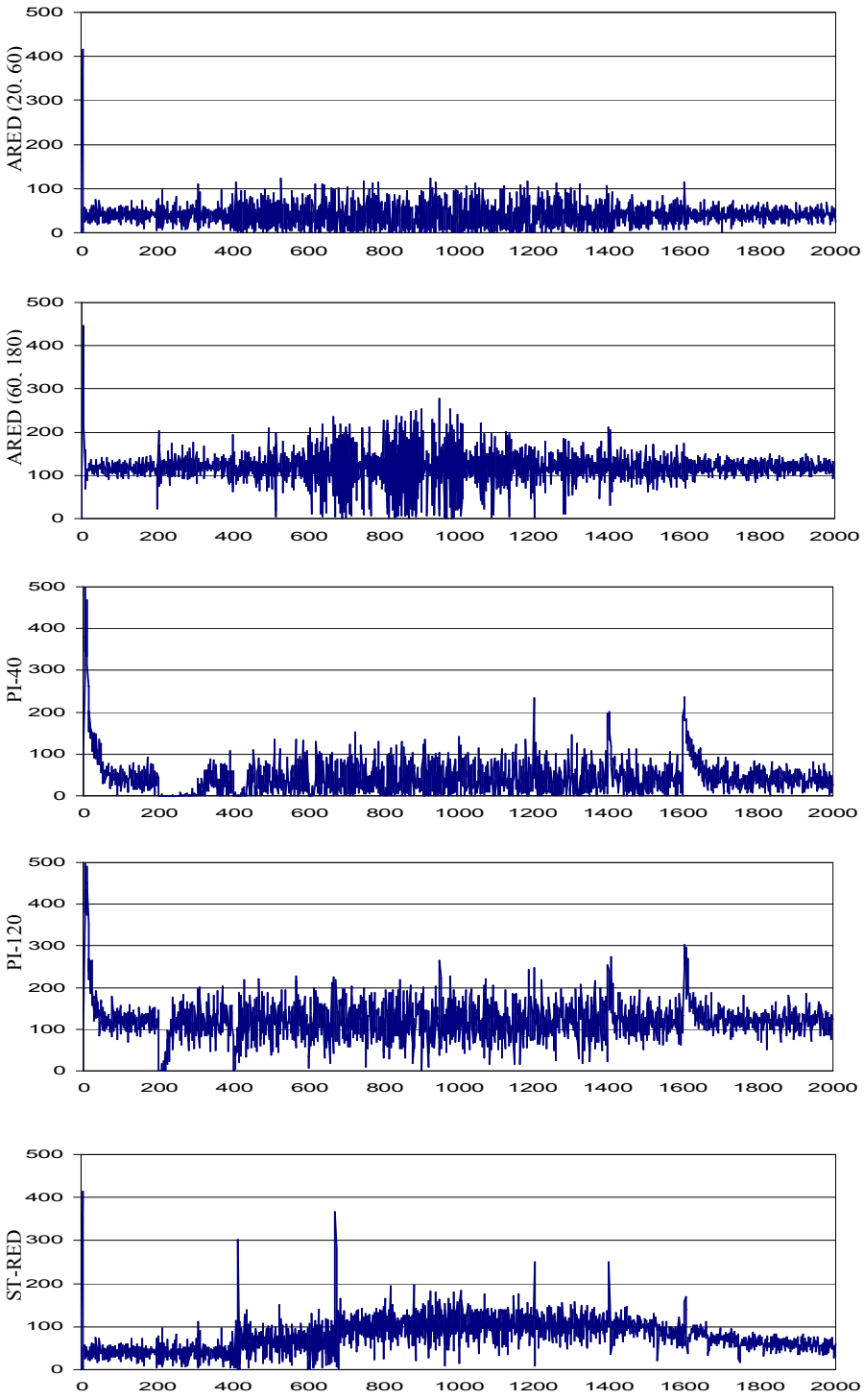


Figure 5.5 Queue size (packets) variations versus time (seconds) under varying bottleneck link capacity [10Mbps-50Mbps]

For PI-40, we see the queue hardly settle down for the first 400 seconds. This is because the PI parameters which can stabilize the queue under high link capacity become too conservative when the capacity is small. So does PI-120. On the other hand, the oscillations of PI-120 are more benign than PI-40.

As the link capacity increases, ST-RED can detect large oscillations and increases average queue size correspondingly. The figure shows that its average queue size increases from 40 packets to the highest about 120 packets. As the link capacity decreases, ST-RED can detect small oscillations and decreases the average queue size correspondingly. Observe that the average queue size returns to about 50 packets when link capacity returns to 10Mbps. During the process ST-RED maintains a stable queue, as well as provide lower queuing delay than ARED (60, 180) and PI-120 most of the times.

Table 5-2 Summary Statistics for all Designs under Varying Bottleneck Link Capacity

	Throughput ($\times 10^9$bytes)	Loss rate	Average queue size	Average queue deviation
ARED (20,60)	6.47	2.40%	39.2	24.1
ARED (30,90)	6.48	2.26%	59.4	29.2
ARED (40,120)	6.49	2.14%	77.8	32.6
ARED (50,150)	6.49	2.02%	99.6	31.6
ARED (60,180)	6.50	1.92%	118.5	34.1
PI-40	6.36	2.60%	42.9	36.8
PI-60	6.43	2.42%	61.7	40
PI-80	6.46	2.29%	82.1	42.4
PI-100	6.48	2.16%	101.6	42.8
PI-120	6.49	2.05%	121.6	43.5
ST-RED	6.50	2.21%	77.1	27

Since link capacity is not a constant in this experiment, throughput is used as a metric instead of link utilization. In Table 5-2, ST-RED provides the highest throughput of 6.5×10^9bytes among all schemes and its average queue deviation of 27 packets is near the smallest value 24.1 of ARED (20, 60). Compared with schemes with smaller average queue size, ST-RED has lower loss rate and higher throughput. ST-RED still provides one of the best performances in this scenario with varying link capacity.

5.5.3 Varying the number of FTP flows

In this experiment the performance of each scheme is checked under different numbers of FTP flows. We start 30 FTP flows at 0, and then increase it to 50, 100, 150 and 200 at the 200th second, 400th second, 600th second and 800th second. Then the number of FTP flows decreases to 150, 100, 50 and 30 at 1000th second, 1200th second, 1400th second and 1600th second, respectively. The round-trip propagation delay is set unchanged at 100ms, and the bottleneck link capacity is 10Mbps. Figure 5.6 depicts the instantaneous queue size of each scheme.

As the parameters of PI controllers are conservative with small *d* and low link capacity, the figure shows that their responsiveness is sluggish and the amplitude of oscillations is relatively large, although the queue can finally be stabilized. For ARED (20, 60) and ARED (60, 180), a stable queue can be seen all the time because the number of FTP flows has little influence on the queue oscillations and each ARED scheme is good at controlling the queue size at a given equilibrium point. Since no large or small change can be detected, the queue sizes of ST-RED keep stable at the equilibrium point of 40 packets all the time. Figure 5.6 and Table 5-3 show that ST-RED has the same performance as ARED (20, 60) with very high link utilization of 99.91%, the smallest average queue size of 40 packets, and the smallest average queue deviation of 14.9. In general, ST-RED shows one of the best performances when traffic load changes.

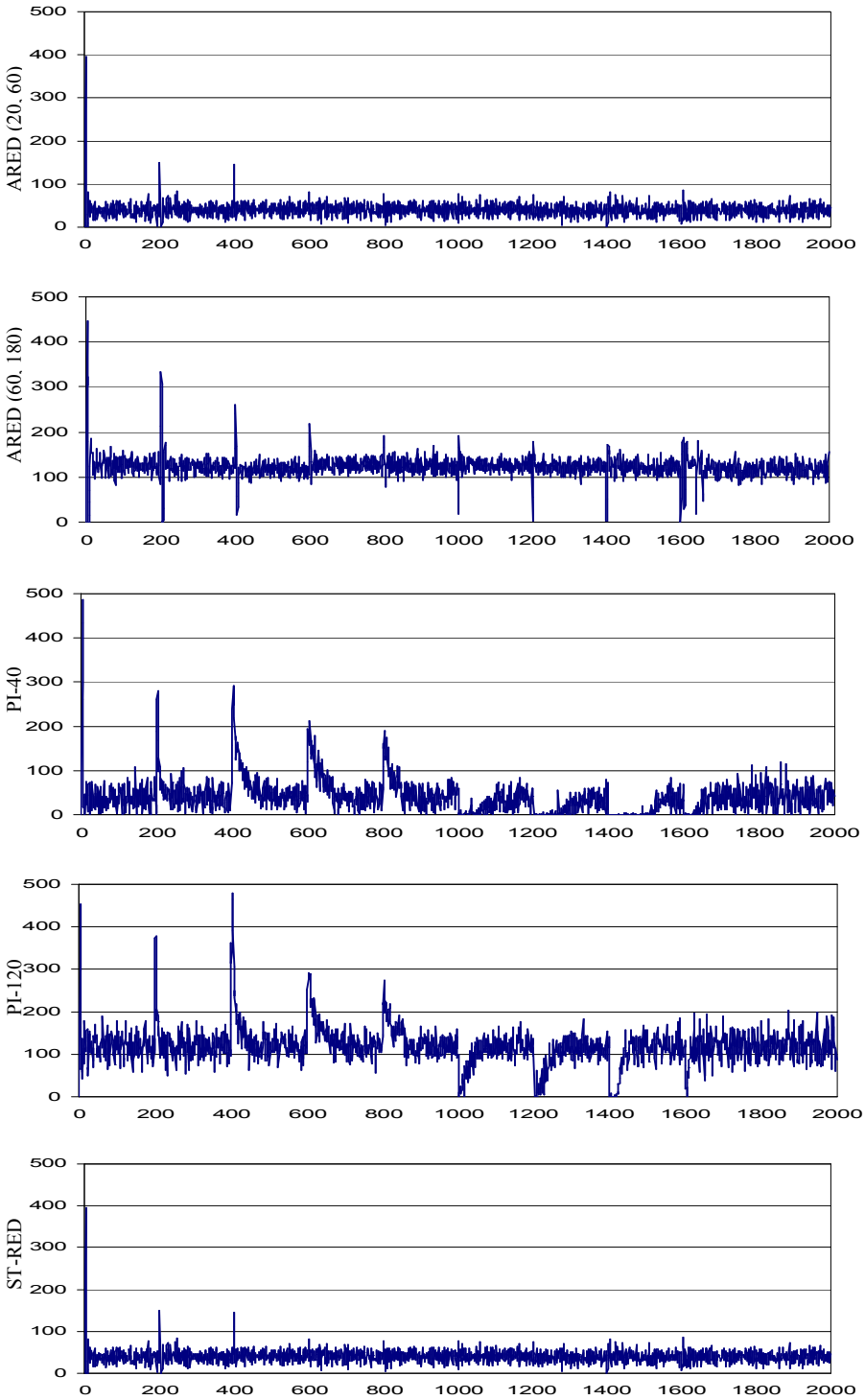


Figure 5.6 Queue size (packets) variations versus time (seconds) under varying number of FTP flows [30-200]

Table 5-3 Summary Statistics for all Designs under Varying Number of FTP Flows

	Link Utilization	Loss rate	Average queue size	Average queue deviation
ARED (20,60)	99.91%	6.93%	40.0	14.9
ARED (60,180)	99.95%	5.45%	122.3	21.7
PI-40	97.09%	7.54%	39.4	32.3
PI-120	99.74%	5.89%	119.2	39.7
ST-RED	99.91%	6.93%	40	14.9

5.5.4 Varying short HTTP sessions

The fourth experiment is run in order to evaluate the performance of all schemes under changing numbers of HTTP flows. Each HTTP session repeatedly makes short file transfers. Between two consecutive transfers, there is a 'think time' that starts after the last byte of the first file has been acknowledged. The transfer size is exponentially distributed with a mean of twelve 1 KB-packets. The think time is exponentially distributed with a mean of 500ms. 1 HTTP sessions are started at 0, and then 1 HTTP sessions are added every 100 seconds until there are 10 HTTP sessions during 900-1000 seconds. In addition, 100 persistent FTP flows exist during the process. The round-trip propagation delay is set unchanged at 100ms, and the bottleneck link capacity is 10Mbps. The instantaneous queue size of each scheme is shown in Figure 5.7.

The figure shows the sluggish response of PI controllers with conservative parameters in the presence of varying HTTP sessions. Although queue size keeps stable, the queue deviation is large. For ARED (20, 60) and ARED (60, 180), the queue size can keep stable with small deviation because short HTTP sessions hardly impact on the oscillations of the queue size. Since no large or small change can be detected,

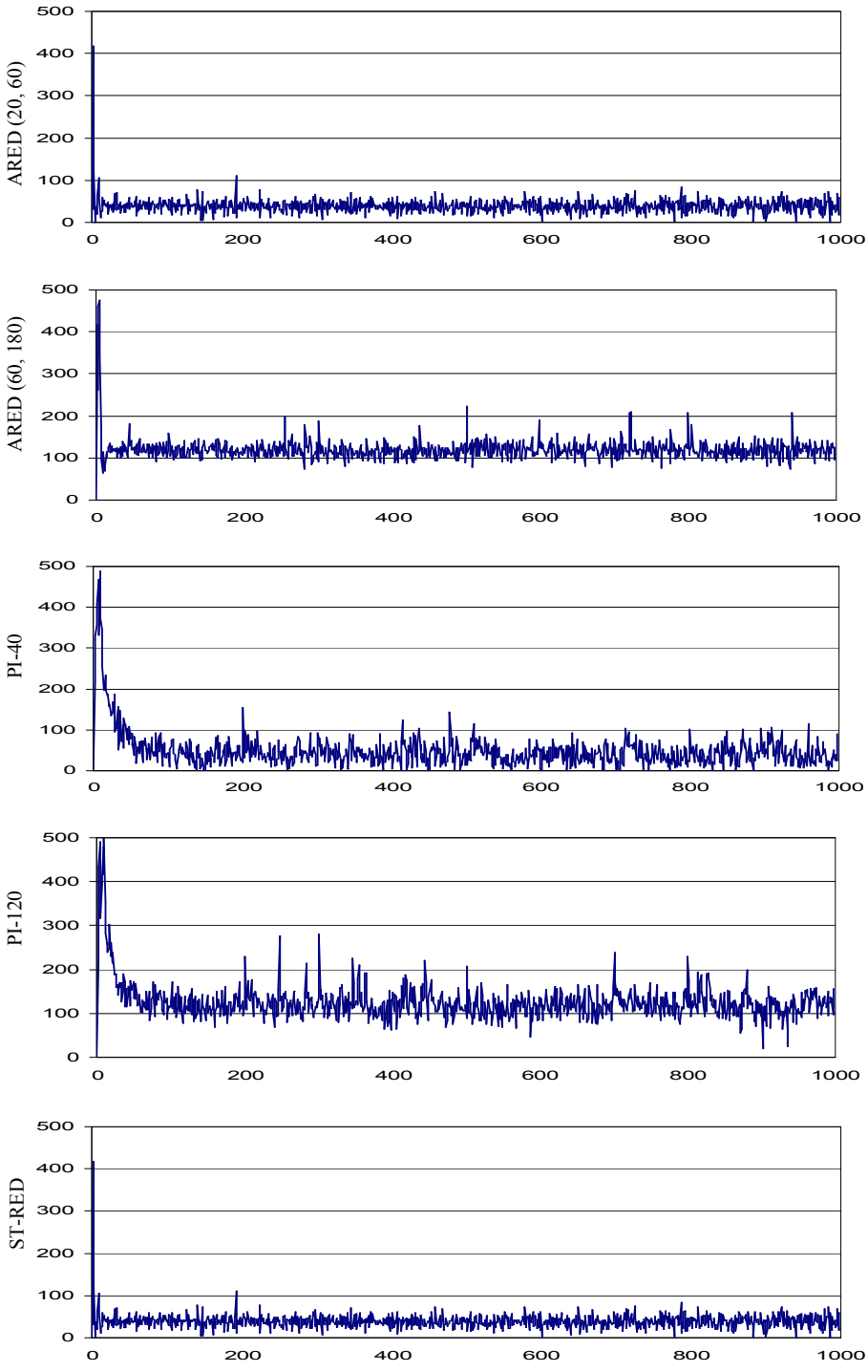


Figure 5.7 Queue size (packets) variations versus time (seconds) under varying HTTP sessions

Figure 5.7 and Table 5-4 show that ST-RED has the same performance as ARED (20, 60) with very high link utilization of 99.92%, the smallest average queue size of 39.1packets, and the smallest average queue deviation of 16.9. In general, ST-RED shows one of the best performances once again when HTTP sessions vary.

Table 5-4 Summary Statistics for all Designs under Varying HTTP sessions

	Link Utilization	Loss rate	Average queue size	Average queue deviation
ARED (20,60)	99.92%	8.31%	39.1	16.9
ARED (60,180)	99.95%	6.66%	119.7	23.6
PI-40	99.65%	8.96%	48.6	35.8
PI-120	99.94%	7.31%	126.3	36.7
ST-RED	99.92%	8.31%	39.1	16.9

5.5.5 Varying the density of the UDP traffic

In the final experiment the performance of each scheme is investigated with the existence of unresponsive UDP flows. Each UDP flow is an ON/OFF flow. The duration of the ON and OFF states are exponentially distributed with a mean of 1 second. During ON time, each UDP flow transmit with a rate of $\rho \times 10/100$ Mbps, where ρ is the density of the UDP traffic. The number of UDP flows introduced in this experiment is 100. The initial ρ is set to 0.1 at 0, and then is increased to 0.3, 0.5, 0.7 and 0.9 at the 200^{th} second, 400^{th} second, 600^{th} second and 800^{th} second. Then ρ decreases to 0.7, 0.5, 0.3 and 0.1 at 1000^{th} second, 1200^{th} second, 1400^{th} second and 1600^{th} second, respectively. There are 100 persistent FTP flows. The round-trip propagation delay is set unchanged at 100ms, and the bottleneck link capacity is 10Mbps. Figure 5.8 depicts the instantaneous queue size of each scheme.

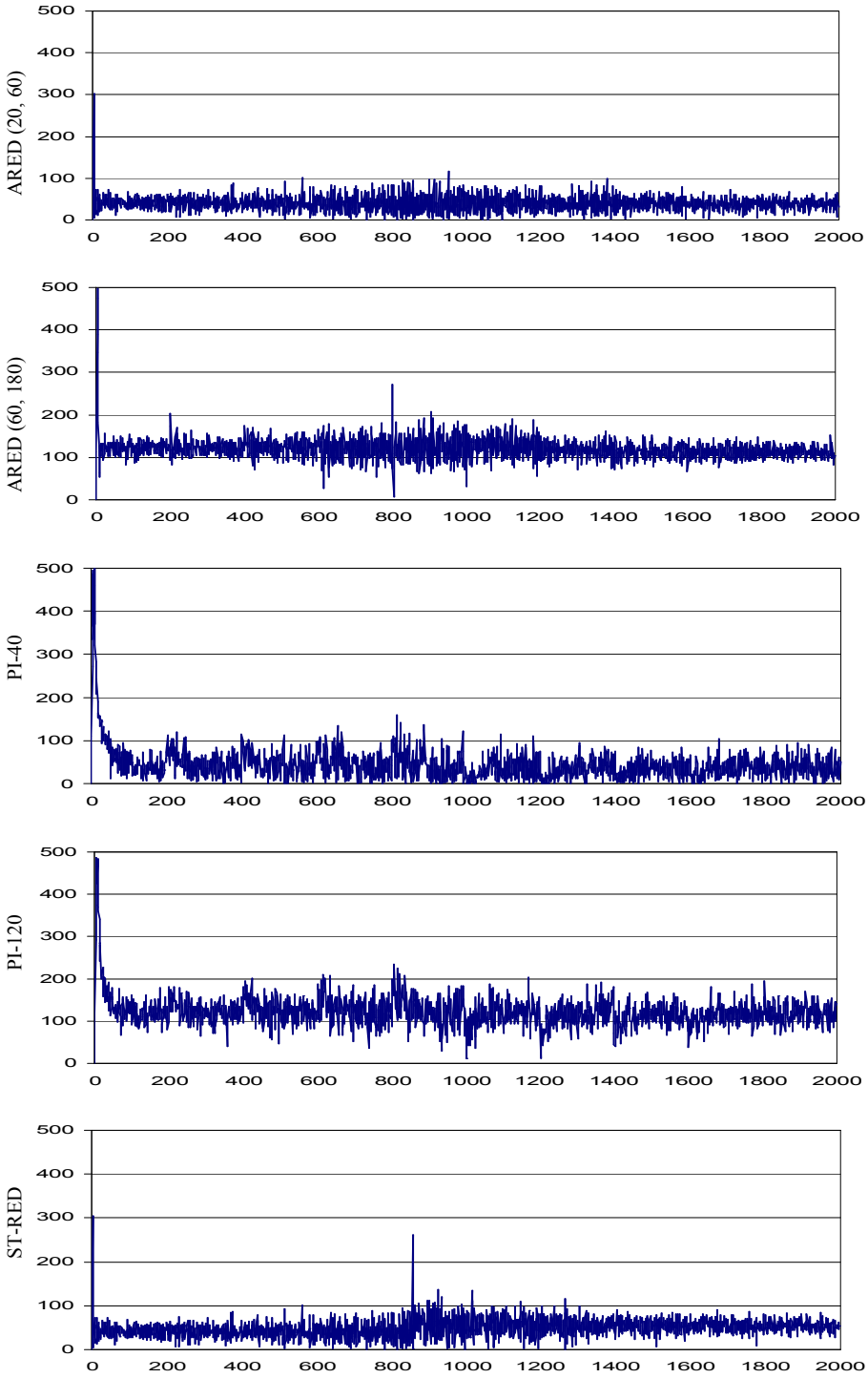


Figure 5.8 Queue size (packets) variations versus time (seconds) under varying UDP traffic density [0.1-0.9]

As the rate of the UDP flows increase, a bit more severe queue oscillations are observed in ARED (20, 60) and in ARED (60, 180). On the other hand, the oscillations introduced by UDP flows is not as significant as that introduced by link capacity and round-trip propagation delay, as shown in Figure 5.4 and Figure 5.5. As for PI-40 and PI-120, these UDP flows result in more severe oscillations and the PI controllers take a long time for the queue to settle down. On the other hand, ST-RED respond to the large oscillations and its parameters are changed correspondingly. Table 5-5 shows that ST-RED provides very high link utilization of 99.94% and its average queue size of 49.1 packets is small. Its average queue deviation of 18.9 is near the smallest value of 17.8. Compared to ARED (20, 60) and PI-40 which has lower average queue size, ST-RED provide higher link utilization and lower loss rate. The average queue size of ST-RED is much lower that of ARED (60, 180) and PI-120. So ST-RED performs very well under varying UDP flows.

Table 5-5 Summary Statistics for all Designs under Varying UDP Traffic Density

	Link Utilization	Loss rate	Average queue size	Average queue deviation
ARED (20,60)	99.92%	10.28%	40.0	17.8
ARED (60,180)	99.96%	8.12%	119.6	23.1
PI-40	99.57%	10.83%	43.5	30.9
PI-120	99.96%	8.64%	122.0	32.2
ST-RED	99.94%	10.05%	49.1	18.9

5.6 Conclusions

This Chapter has analyzed the reason for queue oscillations and thus the change in network scenarios such as round-trip time and link capacity can be inferred from the change in the variance of queue oscillations. Based on this analysis, a ST-RED is de-

veloped for systematically tuning $p_{\max}$ in response to the value of the average queue size to control the equilibrium point of the queue size, and tuning other RED parameters in response to the variance of the queue size to control the stability of TCP/RED systems. Thus, four key RED parameters are adapted to constantly achieve desirable performance under a wide range of uncertainties in network conditions. The algorithm is simple and easy to implement. Simulation results demonstrate that ST-RED can constantly provide high link utilization and keep queuing delay as low as possible in extensively changing network scenarios. The queue deviation of ST-RED is also very small. Compared with ARED and PI controller, ST-RED can provide best performance in almost all varying network scenarios. As RED works well only in conjunction with packet loss, the best performance AQM with ECN need to be studied. So the algorithm for the parameter tuning of PI controller is proposed in Chapter 6.

CHAPTER 6
Statistical Tuning PI Controller in Dynamic network Scenarios

6.1 Introduction

Internet routers use buffers to accommodate packets during congestion. However, buffers cause queuing delay, delay-jitter and other problems. Router buffers are the single biggest contributor to uncertainty in the Internet [4]. Obviously, buffer management plays an important role in the Internet.

By proactively dropping packets before the buffer overflows, Active Queue Management (AQM) aims to stabilize the queue size within a given target and thus to provide both high link utilization and low queuing delay. Chapters 4 and 5 have proposed algorithms to address the parameter tuning problem of RED. Although ARED is ranked as the best scheme without ECN [40], RED or ARED are not the perfect solution in some situations. Its weakness has been discussed in Chapter 2. Chapter 3 has shown many other AQM algorithms proposed in recent years. Among these AQMs, PI and REM perform best when ECN is supported [40]. In addition, PI performs better than REM when ECN is not supported.

As to the PI controller, the parameter tuning algorithms provided in [11][34][60] cannot adjust queuing delay and also depend on the measurement of network parameters. This would make the system complex and increase uncertainty. It is still difficult

to design optimal parameters for existing AQMs such as PI and REM to achieve both desirable steady-state performance and desirable transient response in different network scenarios [10][57].

By investigating the stability analysis in [44], this Chapter develops the mechanism of adapting PI parameters. Then the statistical methodology in Chapter 5 is used to determine the adjustment factor in PI controller a statistical-tuning PI (ST-PI) controller is developed. Theoretical analysis and extensive simulations using the ns-2 simulator demonstrate that ST-PI can constantly stabilize the queue, keep queuing delay as low as possible, has small deviation under widely varying round-trip time, bottleneck link capacity, traffic load, UDP traffic and short HTTP sessions.

The rest of the Chapter is organized as follows. The methodology of ST-PI is introduced in Section 6.2. Section 6.3 describes the details of the ST-PI algorithm. We use ns-2 simulations to verify ST-PI in Section 6.4. Finally, conclusions are presented in Section 6.5.

6.2 Methodology

A PI controller calculates packet drop/mark probability p at every sampling interval T_s. According to (3.6), we have the following transfer function of a PI Controller:

$$C(s) = \frac{K_p(s+z)}{s} \tag{6.1}$$

where K_p is the proportional gain and z is the PIs zero.

When $C_{red}(s)$ in Figure 4.1 is replaced by $C(s)$, the figure presents the feedback control model of TCP/PI system. According to (4.2) and (6.1), we have the open-loop transfer function of TCP/PI system as

$$L(s) = \frac{K_p \frac{C^2}{2N}(s+z)e^{-sR}}{s(s+\frac{2N}{R^2C})(s+\frac{1}{R})} \tag{6.2}$$

The stability proposition of TCP/PI system in [32][33][60] can be presented as follows.

Stability Proposition: let K_p and z satisfy:

$$K_P = 2\lambda\sqrt{\lambda^2+1}\frac{N}{R^2C^2}$$

$$z = \frac{2N}{R^2C} \tag{6.3}$$

As chosen $\lambda \in (0,0.85)$ [33], then the linear TCP/PI system in Figure 4.1 is stable. See proof in [33].

Thus the loop's unity gain crossover frequency is

$$\omega_g = \frac{\lambda}{R} \tag{6.4}$$

At ω_g the loop phase is

$$\angle L(j\omega_g) = -90° - \frac{180}{\pi}\lambda - \arctan\lambda > -180° \tag{6.5}$$

From (6.3) we see that a larger K_p is provided by a larger λ. So a larger w_g can be obtained according to (6.4). This means a faster system response. On the other hand, we see that a smaller K_p can provide more phase margin according to (6.3)-(6.5).

Like AP-RED, the rationale behind setting q_{ref} as a fraction of bandwidth-delay product is to accommodate bursty traffic. The rationale behind setting K_p is to keep system stable while providing fast response.

Consider initial parameters K_{p0} and z_0 that satisfy stability proposition (6.3) and stabilize the queue size around a reference value of q_{ref0} under the initial network scenario of N_0, R_0 and C_0. Assume $N \equiv N_0$. When network scenario varies to any $R \le R_0$ and any $C \le C_0$, the PI parameters can be tuned as follows:

$$q_{ref} = kq_{ref0}$$

$$K_p = \frac{1}{k^2} K_{p0} \tag{6.6}$$

where $k = k_r k_c$; $k_r = \frac{R}{R_0}$; $k_c = \frac{C}{C_0}$.

Proof:

From (6.2)-(6.5), we have the loop gain and phase at the unity gain crossover ω_{g0} under N_0, R_0 and C_0 as

$$\left|L(j\omega_{g0})\right| = \frac{K_{p0}\dfrac{C_0^2}{2N_0}}{\omega_{g0}\left|j\omega_{g0} + \dfrac{1}{R_0}\right|} = 1 \tag{6.7}$$

$$\angle L(j\omega_{g0}) = -90° - \frac{180}{\pi}\omega_{go}R_0 - \arctan\omega_{go}R_0 > -180° \tag{6.8}$$

Given $R \le R_0$ and $C \le C_0$, from (6.6) we have

$$k_r \le 1 \text{ and } k_c \le 1 \tag{6.9}$$

At $\omega = \frac{\omega_{g0}}{k_r}$, from (6.2) we have the loop gain

$$\left|L(j\omega)\right| = \frac{\dfrac{K_{p0}}{k_r^2 k_c^2}\dfrac{k_c^2 C_0^2}{2N_0}}{\dfrac{\omega_{g0}}{k_r}\left|j\dfrac{\omega_{g0}}{k_r} + \dfrac{1}{k_r R_0}\right|} \cdot \frac{\left|j\dfrac{\omega_{g0}}{k_r} + z_0\right|}{\left|j\dfrac{\omega_{g0}}{k_r} + \dfrac{2N_0}{k_r^2 k_c R_0^2 C_0}\right|}$$

$$= \frac{K_{p0}\dfrac{C_0^2}{2N_0}}{\omega_{g0}\left|j\omega_{g0} + \dfrac{1}{R_0}\right|} \cdot \frac{\left|j\omega_{g0} + k_r z_0\right|}{\left|j\omega_{g0} + \dfrac{2N_0}{k_r k_c R_0^2 C_0}\right|}$$

Then from (6.7) and (6.9) we have

$$\left|L(j\omega)\right| \le 1 \tag{6.10}$$

From (6.2) we have the loop phase at ω

$$\angle L(j\omega) = -90° - \frac{180}{\pi}\omega_{g0}R_0 - \arctan(\frac{180}{\pi}\omega_{g0}R_0)$$
$$+ \arctan(\frac{\omega_{g0}}{k_r}\cdot\frac{1}{z_0}) - \arctan\omega_{g0}\frac{k_r k_c R_0^2 C_0}{2N_0}$$

From (6.3) we have

$$z_0 = \frac{2N_0}{R_0^2 C_0}$$

Then from this and (6.8)-(6.9) we have $\angle L(j\omega) > \angle L(j\omega_{g0}) > -180°$

This and (6.10) indicate that the closed-loop system is stable according to the Nyquist stability criterion.

Remark: Equations (6.6) can be used to tune PI parameters in response to any $R \le R_0$ and $C \le C_0$. When round-trip time and link capacity decrease, a large K_p can provide fast response and a small q_{ref} can provide low queuing delay. On the other hand, a large q_{ref} as well as a small K_p can keep the system stable and provide high link utilization when R or C increases within the range $R \le R_0$ and $C \le C_0$. In the face of $R > R_0$ or $C > C_0$, when K_p is adjusted to a smaller value according to equations (6.5) more phase margin is provided. So there is more possibility for the system to keep stable when parameters are adapted in this situation.

6.3 Algorithm

Analysis in [44] indicates that the high gain of TCP $K_{TCP} = \frac{(RC)^2}{2N}$ is mainly responsible for the stability of TCP/AQM systems. Since the order of the bandwidth-delay product RC is much higher than that of the traffic load N in K_{TCP}, round-trip time and link capacity make a significantly greater contribution than traffic load in the determination of the stability of TCP/AQM system. So like the TCP/RED system, the TCP/PI system becomes unstable when the bandwidth-delay product increase, while the im-

pact of traffic load, mice traffic or other factors on queue oscillations is slight. We can design ST-PI algorithm similar to that of ST-RED. The detailed ST-PI algorithm is shown in Figure 6.1.

Initialization:

$\sigma^2 \leftarrow 0$

$g^2 \leftarrow ((h_s \cdot q_{ref})^2 + (h_l \cdot q_{ref})^2)/2$

Every sampling interval:

if $0 < q < 2 \cdot q_{ref}$

Calculate the variance:

$\sigma^2 \leftarrow (1 - \phi_1 \cdot T_s)\sigma^2 + \phi_1 \cdot T_s (q - q_{ref})^2$

if $(1-\gamma)q_{ref} < q < (1+\gamma)q_{ref}$

Calculate the detection function:

$g^2 \leftarrow (1 - \phi_2 \cdot T_s)g^2 + \phi_2 \cdot T_s (q - q_{ref})^2$

if ($g > h_l \cdot q_{ref}$ and $\sigma > h_l \cdot q_{ref}$) or

($g < h_s \cdot q_{ref}$ and $\sigma < h_s \cdot q_{ref}$)

Adjust parameters:

if $g > h_l \cdot q_{ref}$

$k \leftarrow \sigma / (h_l \cdot q_{ref})$

else

$k \leftarrow \sigma / (h_s \cdot q_{ref})$

endif

$q_{ref} \leftarrow k \cdot q_{ref}$ $K_p \leftarrow K_p / k^2$

$g^2 \leftarrow ((h_s \cdot q_{ref})^2 + (h_l \cdot q_{ref})^2)/2$

endif

endif

endif

variables:

σ^2 :variance function of q g^2 : variance detection function of q

k: adjustment factor

Fixed parameters:

ϕ_1 : system time constant; $\phi_2 : 0.25 \cdot \phi_1$; γ : 0.7;

h_l : $\gamma / \sqrt{3}$; h_s : fixed in terms of initial PI parameters

Figure 6.1 Detailed ST-PI algorithm

6.3.1 Calculating the variance and detection function of *q*

Similar to ST-RED, the following GMA function is used to calculate the variance at every sampling interval:

$$\sigma^2 \leftarrow (1 - \phi_1 \cdot T_s)\sigma^2 + \phi_1 \cdot T_s (q - q_{ref})^2 \tag{6.11}$$

The following GMA detection function is used to detect significant changes in the variance of q:

$$g^2 \leftarrow (1 - \phi_2 \cdot T_s) g^2 + \phi_2 \cdot T_s (q - q_{ref})^2 \tag{6.12}$$

6.3.2 Parameter setting

From (6.11) and (6.12) we know that the time constant of the variance function is $\frac{1}{\phi_1}$ and the time constant of the variance detection function is $\frac{1}{\phi_2}$. Similar to ST-RED, we choose the time constant of the variance detection function to be four times the time constant of the variance function. So we have $\phi_2 = 0.25 \cdot \phi_1$. In addition, $\frac{1}{\phi_1}$

should not be less than the time constant of the TCP/PI system to guarantee that steady-state values of the variance come from the steady-state values of the queue size. Our default choice of setting ϕ_1 to the system time constant satisfies this constraint. In practice, the system time constant can be obtained by observation of the system settling time, or the settling time of the instantaneous queue size. For instance, the settling time for the system to settle within 2% of its final value would be four times its time constant [30]. Like ST-RED, h_s needs to be fixed mainly in terms of the initial PI parameters.

6.4 Simulations

Extensive *ns-2* simulations have been conducted to verify the performance of ST-RED and ST-PI against ARED and PI controller with varying round-trip times, general distribution of round-trip times, link capacities, number of long-lived TCP flows, number of reverse-path traffic, number of HTTP sessions and density of UDP traffic, with the support of ECN or without ECN. The simple dumbbell topology used in these simulations is shown in Figure 6.2. In this section, some of the results are presented to demonstrate the effectiveness of ST-RED and ST-PI controller under widely changing network conditions when ECN is supported.

There are 300 FTP flows and 50 HTTP sessions in each simulation except those simulations with varying traffic load. Each HTTP session repeatedly makes short file transfers. Between two consecutive transfers, there is a 'think time' that starts after the last byte of the first file has been acknowledged. The transfer size is exponentially distributed with a mean of twelve 1 KB-packets. The think time is exponentially distributed with a mean of 500ms. The reverse-path traffic in each simulation consists of 50 persistent FTP flows. The bottleneck link capacity is 15 Mbps except in the simulations with varying bottleneck link capacity. The propagation delay *d* is set to values uniformly distributed in (1, 20) ms to mimic realistic propagation delays at access links except those simulations with varying round-trip time and general distribution of

round-trip time. This is because most propagation delays at the access link are less than 20 ms [16]. Although network scenarios change significantly, ARED and PI controller can just choose a set of conservative parameters for a specific scenario. This provides a trade-off between high link utilization and low queuing delay for dynamic network conditions. We design the parameters of ARED and the PI controller for a scenario with 300 FTP flows passing through a 15Mbps bottleneck link with round-trip time 400 ms. The corresponding parameters for ARED are $\min_{th} = 75$, $\max_{th} = 225$, $w_q = 5*10^{-5}$ and interval =2s. Thus q_{ref} of ARED is 150 packets. The parameters for PI are a_= $7.288*10^{-5}$, b_= $7.264*10^{-5}$ that satisfy (6.6) with $q_{ref} = 150$ packets and sampling frequency $\frac{1}{T_s}$ is set to 170HZ. ST-RED uses the same parameters as ARED with $h_s = 0.04/\sqrt{3}$. ST-PI uses the same parameters as PI controller with h_s set to $0.45/\sqrt{3}$ and ϕ_1 set to 1/7. We check the performance of each scheme by changing network variables round-trip time, bottleneck link capacity, traffic load and UDP traffic density. Their performance under general distribution of round-trip time is also investigated. The packet size is 500 bytes. The buffer size is set to 1000 packets which is sufficiently large for the following simulations.

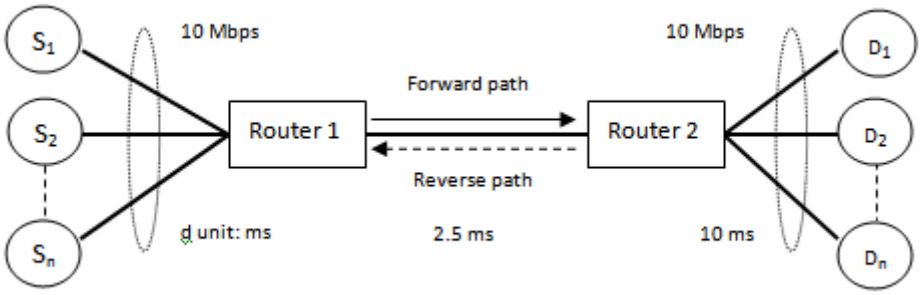


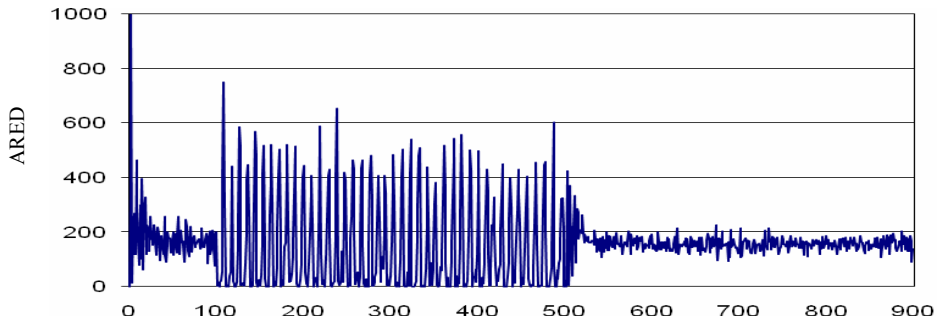
Figure 6.2 Network topology for simulations

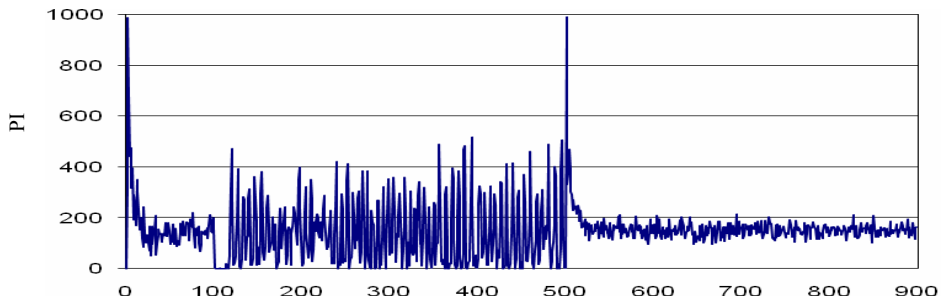
6.4.1 Varying round-trip time

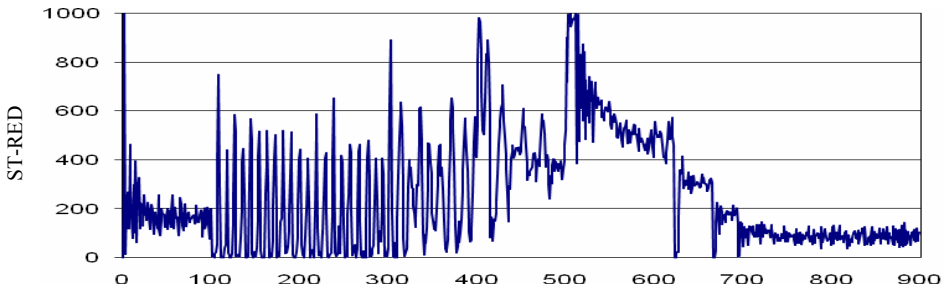
This experiment aims to investigate the performance of each scheme when the round-trip time varies. We vary the round-trip time by changing propagation delay d. The initial propagation delays d at the source links are uniformly distributed in (90, 150) ms. At the 100^{th} second, d increases to values uniformly distributed in (630, 900) ms. At the 500^{th} second, d decreases to values uniformly distributed in (1, 20) ms. We change the seed of the uniform variable d in *ns-2* for random values seven times. Thus the experiments have been done with different random number setting of d for seven times. Figure 6.3 plots the instantaneous queue size for each scheme in one of the seven times. The steady-state performance of each scheme are measured during 400-500 seconds for d = (630, 900) ms and during 800-900 seconds for d = (1, 20) ms. The statistical metric range in seven times is shown in Table 6-1.

For ARED and PI controller the queue oscillations are benign from 0 to 100 seconds because the effective R is near initial design scenario with R = 400 ms. Between 100 seconds and 500 seconds the queue oscillations become severe, corresponding to R much greater than 400 ms. After 500 seconds the queue oscillations reduce significantly as d decreases to small values.

When deterministic oscillations appear after 100 seconds ST-RED and ST-PI can detect significant changes in round-trip time and adjust their parameters correspondingly. After adjustment of parameters, the stability proposition of RED is still satisfied and large phase margin is provided for ST-PI controller. Observe that their queue sizes increase and oscillations become less severe. During 400-500 seconds we mainly consider link utilization because the stability of the system is the most important factor under large round-trip times. See in Table 6-1 that ST-RED and ST-PI provide higher link utilization than ARED and PI controller in all seven times of experiments, giving a confidence value of better than 99% that ST-RED and ST-PI provide higher link utilization than ARED and PI controller. When d decrease to (1, 20) ms after 500 seconds, ST-RED and ST-PI detect small oscillations and reduce their queue sizes


ARED
1000
800
600
400
200
0
0
100
200
300
400
500
600
700
800
900


PI
1000
800
600
400
200
0
0
100
200
300
400
500
600
700
800
900


ST-RED
1000
800
600
400
200
0
0
100
200
300
400
500
600
700
800
900

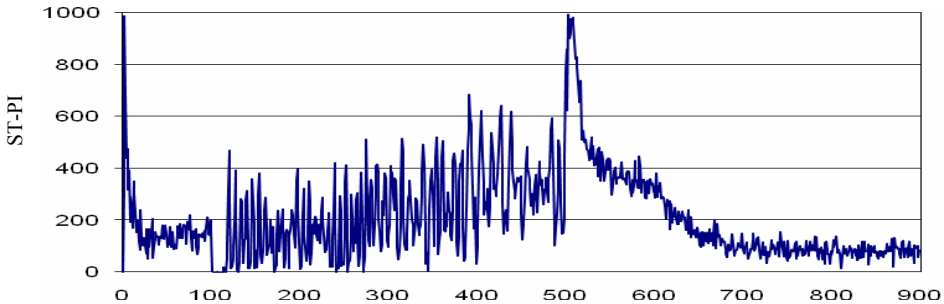

ST-PI
1000
800
600
400
200
0
0
100
200
300
400
500
600
700
800
900

Figure 6.3 Queue size (packets) variations versus time (seconds) under varying round-trip time

Table 6-1 Summary Statistics for all Designs under Varying Round-trip Time

	Time	Link Utilization	Loss rate	Average queue size	Standard queue deviation
ARED	400-500s	92.59%-94.07%	1.03%-1.96%	142.9-159.4	158-204.2
	800-900s	100%	12.74%-13.1%	152.4-156.6	20.3-21.9
PI	400-500s	95.9%-97.6%	0.58%-0.87%	144.9-151.5	123-154.7
	800-900s	100%	0.98%-1.59%	144.6-150.9	19.7-29.2
ST-RED	400-500s	98.36%-99.85%	0.66%-0.91%	348.1-544.3	176.5-283.6
	800-900s	99.93%-100%	13.71%-14.29%	67.9-90.7	22.6-31.2
ST-PI	400-500s	99.29%-99.98%	0.54%-0.82%	296.6-360	125.3-174.4
	800-900s	100%	0.84%-1.11%	78.2-87.3	20.8-25.8

correspondingly. This time we mainly consider average queue size because each scheme can provide very high link utilization. We see in Table 6-1 that the queue sizes

of ST-RED and ST-PI eventually stabilize to values much less than those of ARED and PI controller in all seven times of experiments. So this gives a confidence value of greater than 99% that ST-RED and ST-PI provide smaller queue size than ARED and PI controller.

6.4.2 General distribution of round-trip time

In the second experiment we investigate the performance of each scheme under a wide range of round-trip times. This general distribution of round-trip time is obtained by setting the propagation delay d uniformly distributed in (10, 5000) ms. We change the seed of the uniform variable d for random values seven times. Thus the experiments have been done with different random number setting of d for seven times. Figure 6.4 plots the queue size of each scheme in one of the seven times. The performance ranges in seven times of experiments during 500-600 seconds are shown in Table 6-2.

The queue oscillations of ARED and PI controller are malignant because their conservative parameters are not suitable for this general distribution. On the other hand, ST-RED and ST-PI automatically increase their q_{ref} and adjusts other parameters in response to this general distribution. Figure 6.4 displays that ST-RED and ST-PI can stabilize the queue size at the new reference value. In this experiment we mainly consider link utilization because the stability of the system is the most important factor in this network scenario. We see in Table 6-2 that ST-PI and ST-RED provide higher link utilization than PI controller and ARED in all seven times of experiments. This gives a confidence value of better than 99% that ST-RED and ST-PI provide higher link utilization than ARED and PI controller.

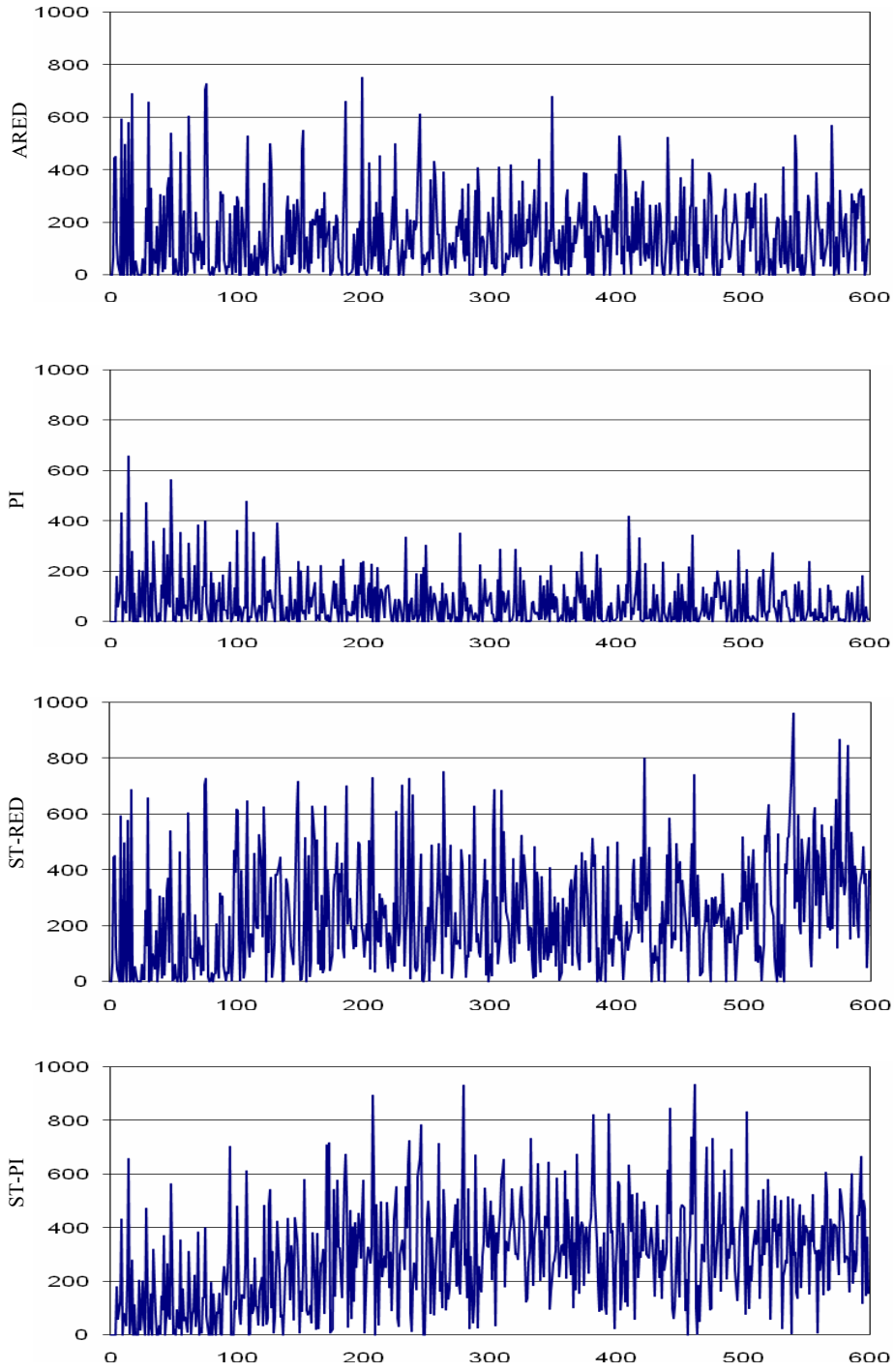


Figure 6.4 Queue size (packets) variations versus time (seconds) under general distribution of round-trip time

Table 6-2 Summary Statistics for all Designs under General Distribution of Round-trip Time

	Link Utilization	Loss rate	Average queue size	Standard queue deviation
ARED	95.6%-97.68%	0.07%-0.4%	131-164.6	122.4-160.4
PI	87.65%-94.32%	0.08%-0.15%	42.9-70.7	56.6-87.8
ST-RED	98.62%-99.83%	0.04%-0.15%	233.6-362.3	154-203.8
ST-PI	98.8%-99.9%	0.07%-0.56%	205.5-487.5	149.9-286.5

6.4.3 Varying bottleneck link capacity

The objectives of the third experiment are to investigate the performance of each scheme under different bottleneck link capacities. The initial link capacity is 15Mbps. It decreases to 5 Mbps at the 200^{th} second and return back to 15 Mbps at the 400^{th} second. We change the seed of the uniform variable *d* for random values seven times. Thus the experiments have been done with different random number setting of *d* for seven times. Figure 6.5 reports the instantaneous queue size of each scheme in one of the seven times. The steady-state performances under changing link capacities are measured during 100-200 seconds, 300-400 seconds and 500-600 seconds. The statistical metric range in seven times of experiments is shown in Table 6-3.

ARED stabilizes the queue size around its reference value of 150 packets and can quickly return back to steady state after the link capacity changes. Although PI controller can also keep the queue size stable around its reference value of 150 packets, it responds slowly to the change in link capacity. This is because its parameters are too conservative when round-trip time is small.

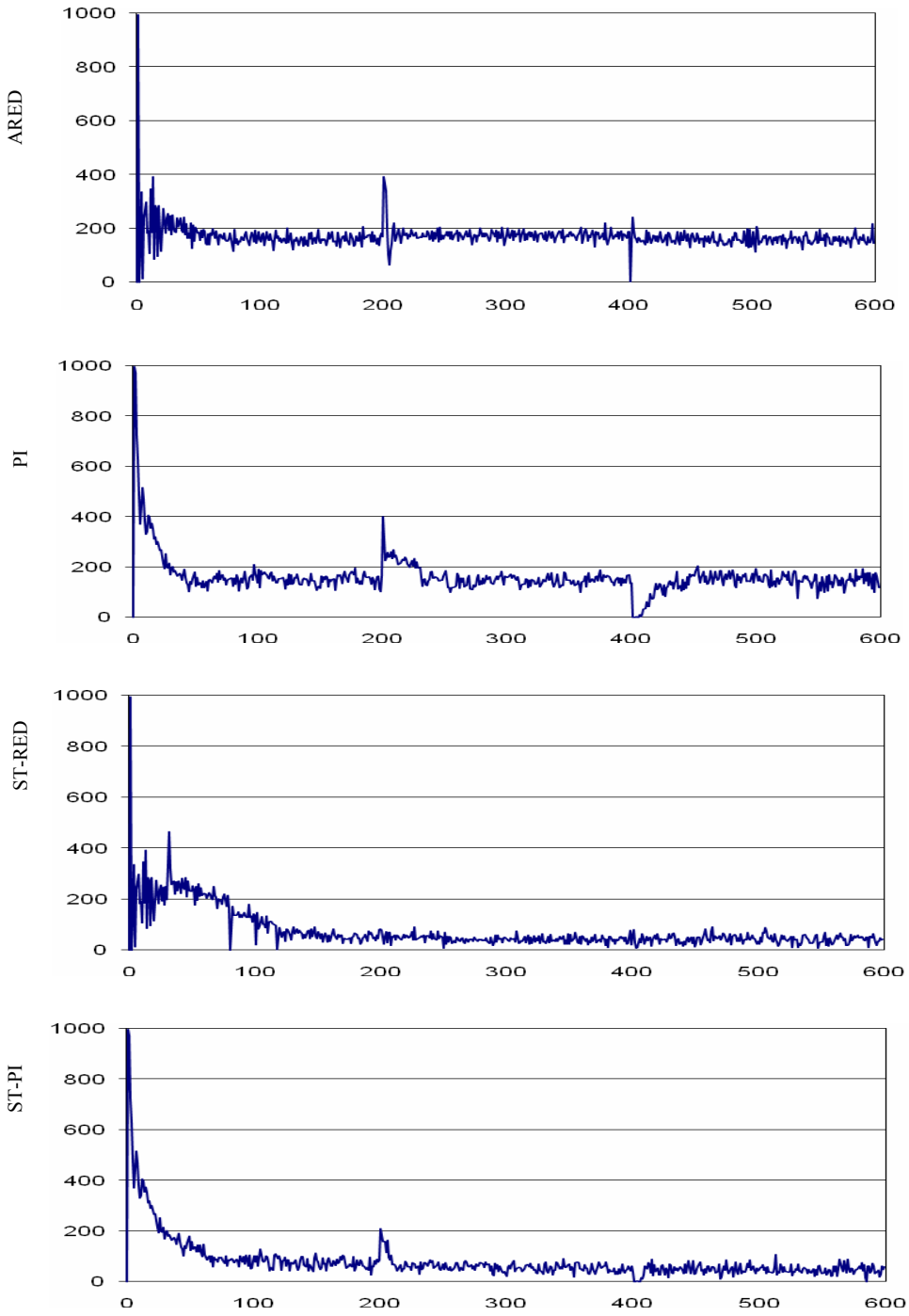


Figure 6.5 Queue size (packets) variations versus time (seconds) under varying bottleneck link capacity

Table 6-3 Summary Statistics for all Designs under Varying Bottleneck Link Capacity

	Time	Link Utilization	Loss rate	Average queue size	Standard queue deviation
ARED	100-200s	100%	19.29%-19.57%	158.2-160.6	17.5-18.7
	300-400s	100%	23.8%-24.25%	174.3-175.4	15.1-18.6
	500-600s	100%	19.01%-19.28%	158.2-160.9	16.6-18.6
PI	100-200s	100%	0.91%-1.06%	145.3-150.7	18.8-23.6
	300-400s	100%	1.91%-2.06%	147.0-149.1	14.6-16.9
	500-600s	100%	0.9%-1.04%	147.3-153.1	18.7-25.1
ST-RED	100-200s	99.92%-99.99%	20.58%-21.05%	54.1-65.5	16-26.1
	300-400s	99.98%-100%	27.84%-28.13%	39.4-44.7	11.2-15.2
	500-600s	99.90%-99.98%	20.96%-21.23%	34.5-43.1	14.8-17.8
ST-PI	100-200s	99.99%-100%	0.78%-0.86%	48.7-74.7	15.9-20
	300-400s	99.99%-100%	1.74%-1.85%	50.4-56.9	13.9-16.7
	500-600s	99.98%-100%	0.82%-0.89%	46.2-55.5	15-17.4

According to the detected small variance corresponding to small *d*, ST-RED and ST-PI reduces the queue size to less than half of its initial reference value of 150 packets. When link capacity decreases to 5 Mbps after 200 seconds, queue oscillations become smaller and thus queue size can be further reduced to about one third of that of ARED and PI. Although the queue oscillations of ST-PI and ST-RED become a little large after link capacity returns back to 15 Mbps after 400 seconds, the variance of the queue size is not too large. Thus their queue sizes are still kept around one third of that of ARED and PI. In this network scenario we mainly consider the metric of average queue size because each scheme can provide very high link utilization of greater than 99.9%. In all seven times of experiments the average queue sizes of ST-PI and ST-RED are significantly less than ARED and PI controller, giving a confidence value of greater than 99% that ST-RED and ST-PI provide smaller queue size than ARED and PI controller. The loss rate of ST-PI is also much less than that of ST-RED and ARED in all seven experiments, providing a confidence value of greater than 99% that ST-PI has lower loss rate than ST-RED and ARED. Although the loss rate of ST-RED is a little higher than ARED, its average queue size is much smaller than ARED. Thus the overall performance of ST-RED is much better than ARED.

6.4.4 Varying traffic load

In this experiment the performance of each scheme is checked under different numbers of FTP flows and different numbers of HTTP sessions. We start 100 FTP flows and 20 HTTP sessions at 0, and increase them to1000 FTP flows and 200 HTTP sessions respectively at 300^{th} seconds. The experiments have been done with different random number setting of *d* for seven times. Figure 6.6 depicts the queue size of each scheme in one of the seven times. The steady-state performance for the first traffic scenario and second traffic scenario are measured during 200-300 seconds and 500-600 seconds respectively. The statistical metric range in seven times of experiments is shown in Table 6-4.

ARED can stabilize the queue size around 150 packets for the first 300 seconds while slightly increase the queue size in the face of heavy traffic load after 300 seconds. PI controller has small oscillations for the first 300 seconds while the oscillations become obvious when traffic load increases after 300 seconds. This is because the parameters of PI controller are conservative with small d = (1-20) ms.

With small round-trip times, the queue sizes of ST-RED and ST-PI reduce to about one third those of ARED and PI controller and keep stable under changing traffic load. See in Table 6-4 that they keep a high link utilization of near 100%. After traffic load changes at 300 seconds, we see that the transient responses of ST-PI and ST-RED are much faster than that of PI controller and slightly better than that of ARED. In this scenario we mainly consider the average queue size because every scheme can provide high link utilization. We also consider the standard queue deviation to study their transient performance under varying traffic load. In all seven times of experiments, the average queue sizes of ST-PI and ST-RED are much less than ARED and PI, giving a confidence value of better than 99% that the average queue size of ST-RED and ST-PI is smaller than ARED and PI controller. The standard queue deviation of ST-PI is also smaller than PI controller in seven times of experiments, giving a confidence value of greater than 99% that ST-PI has a smaller standard queue deviation than PI controller. On the other hand, the loss rate of ST-PI is much less than ST-RED.

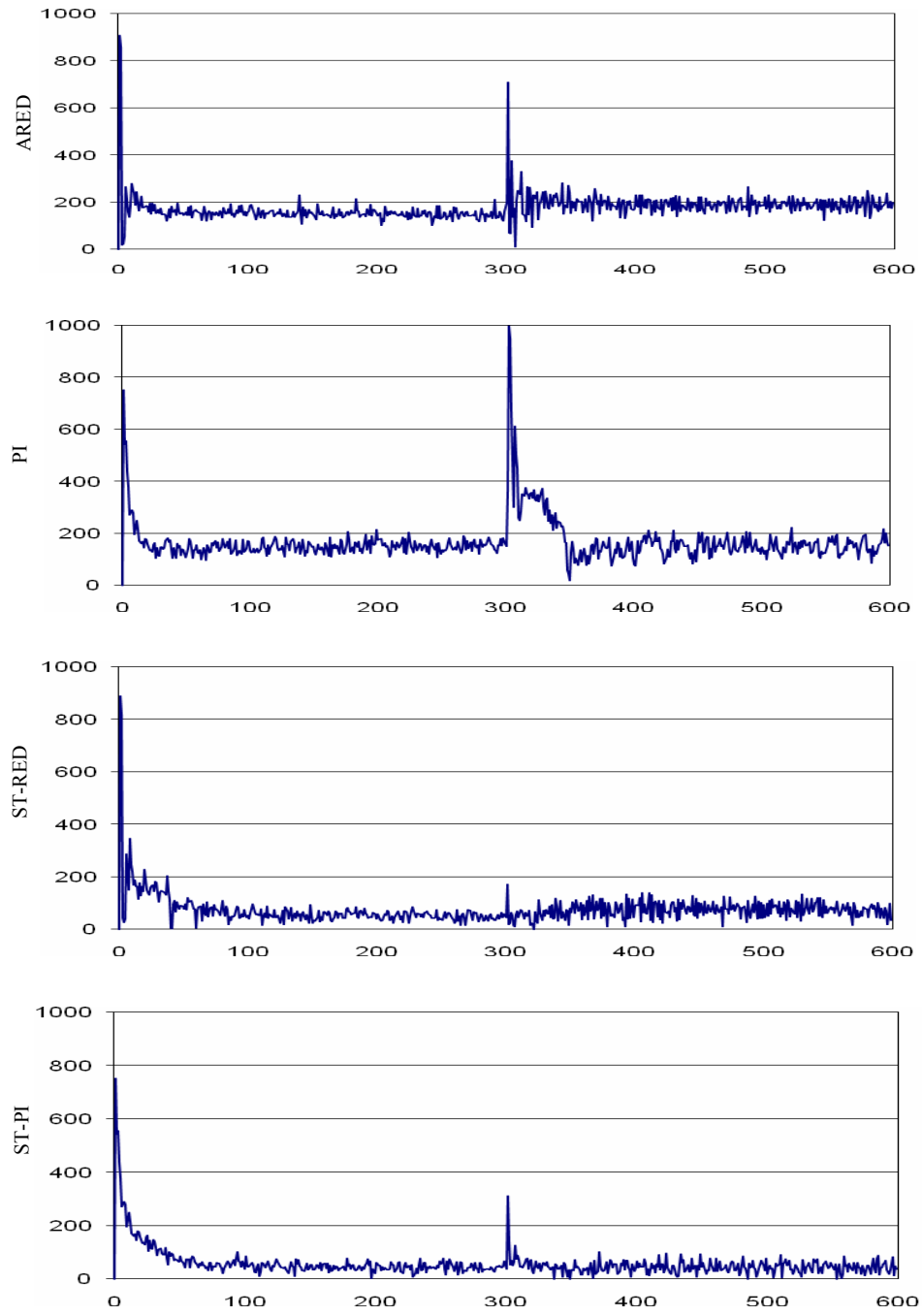


Figure 6.6 Queue size (packets) variations versus time (seconds) under varying traffic load

Table 6-4 Statistics for all Designs under Varying Traffic Load

	Time	Link Utilization	Loss rate	Average queue size	Standard queue deviation
ARED	200-300s	100%	9.63%-10.95%	147.5-153.2	15.6-21.3
	500-600s	100%	26.94%-28.91%	184.5-191.9	21.2-24.8
PI	200-300s	100%	2%-2.66%	145.6-152.4	17.6-22.5
	500-600s	100%	1.17%-1.28%	145.1-153.3	27.8-33.4
ST-RED	200-300s	99.99%-100%	11.36%-12.32%	44.9-53.5	13.5-16.1
	500-600s	99.98%-100%	28.48%-30.54%	47.2-78.9	17.2-29.2
ST-PI	200-300s	100%	1.25%-1.7%	53.8-56.7	12.1-16.8
	500-600s	99.95%-100%	1.67%-2.09%	50.2-55.1	18.2-24.1

6.4.5 Varying the density of the UDP traffic

In the final experiment the performance of each scheme is investigated with the existence of unresponsive UDP flows. Each UDP flow is an ON/OFF flow. The duration of the ON and OFF states are exponentially distributed with a mean of 1 second. During ON time, each UDP flow transmits with a rate of $\rho \times 15/100$ Mbps, where ρ is the density of the UDP traffic. The number of UDP flows introduced in this experiment is 100. The initial ρ is set to 0.1 at 0, and is increased to 0.9 at 300^{th} second.

The seed of the uniform variable *d* has been changed for random values seven times, so the experiments have been done with different random setting of *d* for seven times. Figure 6.7 shows the instantaneous queue size for each scheme in one of the seven times. The steady-state performance of each scheme under different UDP density is measured during 200-300 seconds and 500-600 seconds. The statistical metric range in seven times of experiments is reported in Table 6-5.

The steady-state queue oscillations during 0-300 seconds for ARED and PI controller are small. The oscillations become larger when ρ increases to 0.9, but the queue size for either ARED and PI controller is still stable around their reference values of 150 packets.

The queue sizes of ST-RED and ST-PI decrease to around one third those of ARED and PI controller in the face of small round-trip times. Although queue oscillations of either ST-RED or ST-PI increase after the density of UDP traffic increases to 0.9, the oscillations are not too large. See that the queue sizes of ST-RED and ST-PI are still much smaller than ARED and PI controller in all seven times of experiments. This gives a confidence value of better than 99% that ST-RED and ST-PI have a smaller average queue size than ARED and PI controller. From Table 6-5 we can see that link utilization for all the four schemes are very high. The average queue deviation of ST-RED is less than ARED and the average queue deviation of ST-PI is less than PI controller in the face of high UDP traffic density in all seven times of experiments, giving a confidence value of greater than 99%. This is because the parameters of ST-RED and ST-PI are adjusted to values suitable for the changed scenarios while ARED and PI controller have to use conservative parameters. Compared to ST-RED, ST-PI still has much lower loss rate.

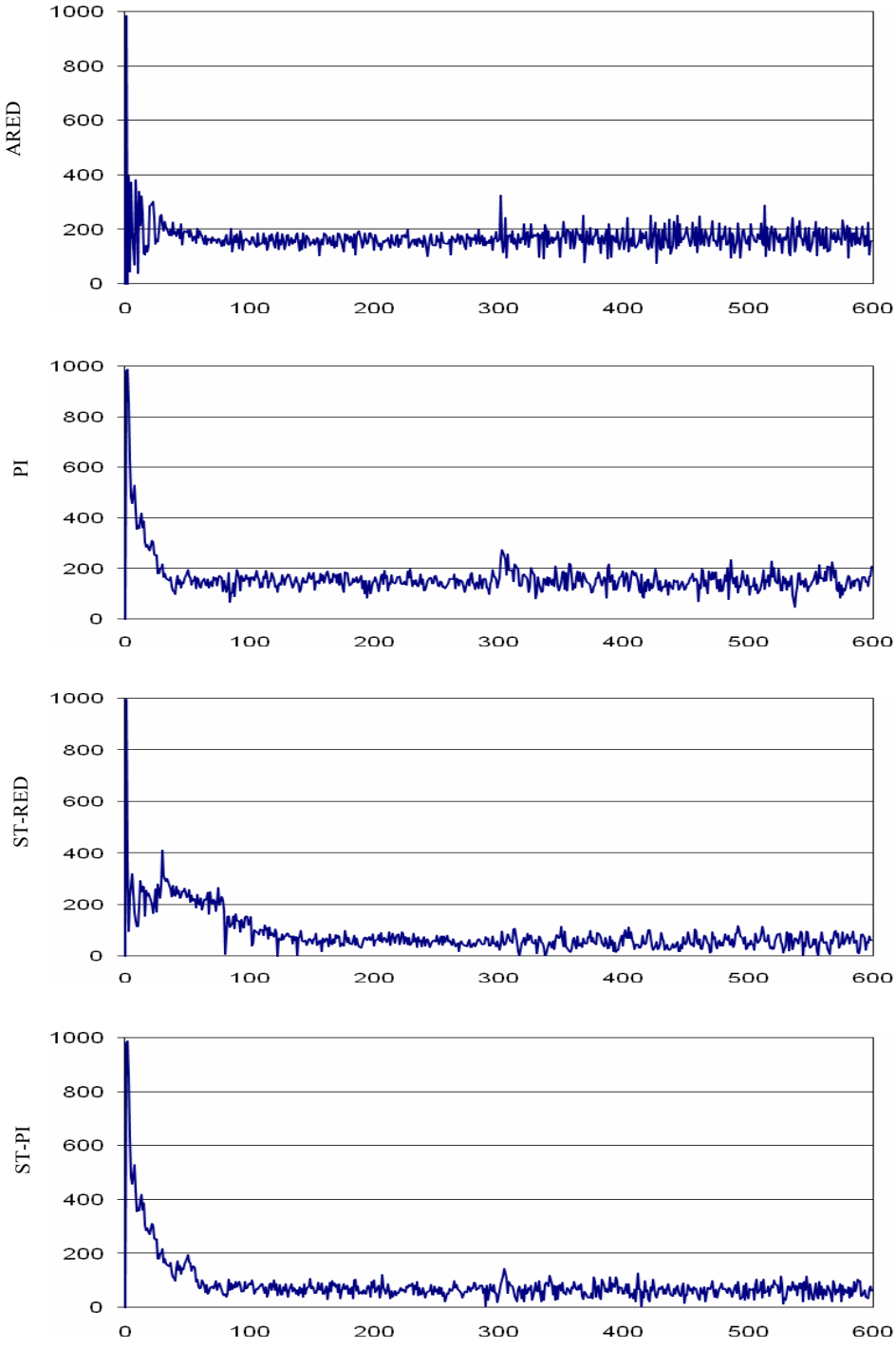


Figure 6.7 Queue size (packets) variations versus time (seconds) under varying UDP traffic density

Table 6-5 Summary Statistics for all Designs under Varying UDP Traffic Density

	Time	Link Utilization	Loss rate	Average queue size	Standard queue deviation
ARED	200-300s	100%	19.52%-19.88%	156.5-159	17.9-19.8
	500-600s	100%	21.77%-22.06%	166.6-172.5	31.2-38.1
PI	200-300s	100%	2.55%-2.71%	148-153	20.3-25.6
	500-600s	100%	15.29%-15.73%	143.7-153.3	25.6-32.3
ST-RED	200-300s	99.88%-100%	21.17%-21.34%	44.5-58.6	15.1-19.8
	500-600s	99.96%-100%	22.26%-23.3%	58.7-83.5	24.7-30.1
ST-PI	200-300s	99.99%-100%	2.62%-2.74%	55.1-73.4	17.5-20.9
	500-600s	99.97%-100%	16.19%-16.5%	53.7-76.1	21.8-23.5

6.5 Conclusions

Based on this analysis that the change in the amplitude of queue oscillations mainly results from the change in round-trip time and link capacity, this chapter has developed a ST-PI algorithm to adapt the parameters of PI controllers to constantly achieve desirable performance under a wide range of uncertainties in network conditions. It is simple and easy to implement. This statistical Tuning algorithm may be used in other AQM schemes. Simulation results demonstrate that ST-PI can constantly provide high

link utilization, fast response, as well as keeping low queuing delay in extensively changing network scenarios such as round-trip time, traffic load, UDP traffic and short HTTP sessions. The transient response and queue deviation of ST-PI in some scenarios like varying traffic load are much better than PI controller. With ECN, ST-PI performs better than existing AQM algorithms. On the other hand, although the assumption that the queue length is a uniform variable, and the default and fixed parameters of ST-RED and ST-PI work well in simulations, they need to be widely verified in practice. In future work we plan to explore the performance of ST-PI and ST-RED under complex network scenarios, including multi-service situations, multiple bottleneck links and realistic networks.

CHAPTER 7
Conclusions

Since RED was proposed in 1993, RED has been widely deployed in commercial routers. However, it is difficult to adapt RED parameters to constantly provide desirable performance in the presence of highly dynamic network traffic. This discourages network administrators to turn RED on. Perhaps motivated by the difficulty, other AQMs have been provided in recent years. Some of them have really improved the performance of original RED and present more stable parameters than RED. But none of them can constantly provide both desirable transient performance and steady-state performance. In particular, they still suffer from low link utilization, high queuing delay or large queue deviation under varying network conditions especially varying round-trip times.

This thesis has focused on solving this key issue related to Internet congestion control. The algorithms proposed in this thesis have demonstrated their abilities of removing parameter sensitivity of some prominent AQMs and of constantly providing desirable performance under changing network scenarios. More specifically, this thesis has:

- Reviewed RED in respect of its algorithm, its parameters, its performance and the implementation. How network conditions determine RED parameters has been

analyzed and how RED parameters impact on network performance has been reviewed. The advantages and disadvantages of RED have been summarized. The stability of RED under different Internet applications has also been investigated. Thus a thorough understanding of RED under Internet traffic has been given.

- Reviewed other AQMs including ARED, Auto-Tuning RED, PI controller, REM, BLUE and AVQ. The characteristics and advantages of these AQMs have been presented. The reasons why these AQMs do not work well in certain situations has been analyzed. Analysis demonstrates that with fixed parameters the performance of these AQMs cannot constantly provide desirable performance.

- Developed a stability proposition for TCP/RED system. It shows why round-trip time and link capacity make much more contribution than traffic load in the determination of the stability of TCP/RED system. According to the stability proposition, AP-RED is proposed to adapting RED parameters to TCP traffic based on measurement of key network variables. AP-RED demonstrates a simple rule about how to tune RED parameters independently without consideration of the interactions among these RED parameters.

- Proposed ST-RED for constantly tuning RED parameters in response to the characteristics of the queue size. ST-RED can adjust RED parameters rapidly to achieve desirable transient and steady-state performance under widely changing network conditions. The algorithm is very simple for implementation. Without ECN, simulations have demonstrated that ST-RED outperforms existing AQMs.

- Proposed ST-PI for removing the sensitivity of PI parameters from different network scenarios. ST-PI can adjust corresponding parameters rapidly to maintain stable performance and keep queuing delay as low as possible in response to the change in network conditions such as round-trip time, link capacity, traffic load, UDP traffic and the number of HTTP sessions. The algorithm is very simple for

implementation. When ECN is enabled, ST-PI can illustrate better performance than existing AQMs.

As the Internet continues to develop, the bandwidth increases and all kinds of new technologies and new services emerge and prosper. Correspondingly new challenges occur. Parameter tuning is an important issue which needs to be addressed in all kinds of academic or industry areas. In future work, some key aspects of this thesis can be extended. They include:

- Some assumptions such as the queue length is a uniform variable, and the default and fixed parameters of ST-RED and ST-PI need to be widely verified in practice. Evaluating the performances of AP-RED, ST-RED and ST-PI under multi-service situations with a more realistic topology. Their performances under realistic network conditions also need to be checked.
- Designing algorithms to solve the challenge to Internet congestion control in the presence of high bandwidth-delay product networks. As bandwidth in current networks has increased significantly, existing AQMs with adaptive parameter tuning need to be evaluated.
- Exploring the possibility to apply Statistical Tuning algorithm to solve parameter-tuning problems related to congestion control in wireless networks and FAST TCP [58].
- Exploring the possibility to apply Statistical Tuning algorithms to solve parameter-tuning issues in other applications like parameter tuning of PID controller.

References

[1] J. Aikat, J. Kaur, and F. D. Smith, “Variability in TCP Round-trip Times,” in Proceedings of Internet Measurement Conference, Florida, USA, pp. 279-284, October 2003.

[2] A. Akella, S. Seshan, and A. Shaikh, “An Empirical Evaluation of Wide-Area Internet Bottlenecks,” in Proceedings of Internet Measurement Conference, Florida, USA, pp. 101-114, October 2003.

[3] M. Allman, V. paxson, and W. Stevens, “TCP Congestion Control,” RFC 2581, April 1999.

[4] G. Appenzeller, I. Keslassy, and N. Mckeown, “Sizing Router Buffers,” in Proceedings of IEEE ACM/SIGCOMM, Oregon, USA, pp. 281-292, August 2004.

[5] S. Athuraliya, V. H. Li, S. H. Low, and Q. Yin, “REM: Active Queue Management,” IEEE Network, vol. 15, No. 3, pp. 48-53, May 2001.

[6] M. Basseville and I. V. Nikiforov, “Detection of Abrupt Changes: Theory and Application,” Prentice-Hall, 1998.

[7] T. Bonald, M. May, and J. Bolot, “Analytic Evaluation of RED Performance,” in Proceedings of IEEE INFOCOM, Tel Aviv, Israel, pp. 1415-1424, March 2000.

[8] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang, “Recommendations on Queue Management and Congestion Avoidance in the Internet,” RFC 2309, April 1998.

[9] R. Cáceres, N. Duffield, A. Feldmann, J. D. Friedmann, A. Greenberg, R. Greer, T. Johnson, C. R. Kalmanek, B. Krishnamurthy, D. Lavelle, P. P. Mishra, J. Rexford, K. K. Ramakrishnan, F. D. True, and J. E. van der Merwe, "Measurement and Analysis of IP Network Usage and Behavior," IEEE Communications Magazine, vol 38, No. 5, pp. 144-151, May 2000.

[10] X.L. Chang and J.K. Muppala, "A Stable Queue-based Adaptive Controller for Improving AQM Performance," Computer Networks, vol. 50, No. 13, pp. 2204-2224, September 2006.

[11] Q. Chen and O. W. W. Yang, "A ST-PI-PP Controller for AQM Router," in Proceedings of IEEE ICC, Paris, France, pp. 2277-2281, June 2004.

[12] W. Chen and S. H. Yang, "The Mechanism for Adapting RED Parameters to TCP Traffic," Computer Communications, vol. 32, No. 13-14, pp. 1525-1530, August 2009.

[13] M. Christiansen, K. Jeffay, D. Ott, and F. D. Smith, "Tuning RED for Web Traffic," IEEE/ACM Transaction on Networking, vol. 9, No. 3, pp. 249-264, June 2001.

[14] S. D. Cnodder, O. Elloumi, and K. Pauwels, "Effect of different packet sizes on RED performance," In Proceedings of the fifth IEEE symposium on computers and communications (ISCC), France, July 2000

[15] A. Dhamdhere, H. Jiang and C. Dovrolis, "Buffer Sizing for Congestion Internet Links," in Proceedings of IEEE INFOCOM, Miami, USA, pp. 1072-1083, March 2005.

[16] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu, "Characterizing Residential Broadband Networks," in Proceedings of Internet Measurement Conference, San Diego, USA, pp. 43-56, October 2007.

[17] W. M. Eddy and M. Allman, "A Comparison of RED's Byte and Packet Modes," Computer Networks, vol. 42, No. 2, pp. 261-280, June 2003.

[18] W. C. Feng, D. D. Kandlur, D. Saha, and K. G. Shin, “A Self-Configuring RED Gateway,” in Proceedings of IEEE INFOCOM, New York, USA, pp. 1320- 1328, Mar 1999.

[19] W. C. Feng, Kang G. Shin, Dilip D. Kandlur, and Debanjan Saha “BLUE Active Queue Management Algorithms,” IEEE/ACM Transactions on Networking, vol. 10, No. 4, pp. 513-528, August 2002.

[20] S. Floyd “Recommendation on Using the “gentle_” Variant of RED,” [Online]. Available: http://www.icir.org/floyd/red/gentle.html

[21] S. Floyd and V. Jacobson, “Random Early Detection Gateways for Congestion Avoidance,” IEEE/ACM Transactions on Networking, vol. 1, No. 4, pp. 397–413, August 1993.

[22] S. Floyd, R. Gummadi, and S. Shenker, “Adaptive RED: An Algorithm for Increasing the Robustness of RED’s Active Queue Management,” [Online]. Available: http://www.icir.org/floyd/red.html

[23] S. Floyd and K. Fall, “Promoting the Use of End-to-End Congestion Control in the Internet,” IEEE/ACM Transactions on Networking, vol. 7, No. 4, pp. 458-472, August 1999.

[24] S. Floyd, “A Report on Recent Developments in TCP Congestion Control,” IEEE Communications Magazine, vol. 39, No. 4, pp. 84-90, April 2001.

[25] S. Floyd, “TCP and Explicit Congestion Notification,” ACM/SIGCOMM Computer Communications Review, vol. 24, pp. 10-23, Oct. 1994.

[26] S. Floyd, “RED: Discussions of Setting Parameters,” Nov. 1997, [Online]. Available: http://www.aciri.org/floyd/REDparameters.txt

[27] S. Floyd, “RED: Discussions of Byte and Packet Modes,” March 1997, [Online]. Available: http://www-nrg.ee.lbl.gov/floyd/REDaveraging.txt

[28] S. Floyd, “Re: RED gateways in byte mode vs. packet mode,” October 2000, [Online]. Available: http://www-nrg.ee.lbl.gov/floyd/REDaveraging.txt

[29] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot, "Packet-Level Traffic Measurements from the Sprint IP Backbone," IEEE Network, vol. 17, No. 6, pp. 6-16, November-December 2003.

[30] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, "Feedback Control of Dynamic Systems," pp. 26-31, Prentice Hall, 2002.

[31] C. Hollot, V. Misra, D. Towlsey, and W. Gong, "A Control Theoretic Analysis of RED," in Proceedings of IEEE INFOCOM, Anchorage, USA, pp. 1510-1519, April 2001.

[32] C. Hollot, V. Misra, D. Towlsey, and W. Gong, "On Designing Improved Controllers for AQM Routers Supporting TCP Flows," in Proceedings of IEEE INFOCOM, Anchorage, USA, pp. 1726-1734, April 2001.

[33] C. Hollot, V. Misra, D. Towlsey, and W. Gong, "Analysis and design of controllers for AQM routers supporting TCP Flows," IEEE Transactions on Automatic Control, vol. 47, No. 6, pp 945-959, June 2002.

[34] Y. Hong and O. W. W. Yang, "Using interval phase margin assignment to self-tune a PI AQM controller for TCP traffic," Telecommunication Systems, vol. 36, No. 4, pp. 161-171, December 2007.

[35] G. Iannaccone, C. Diot, I. Graham, and N. Mckeown, "Monitoring Very High Speed Links," in Proceedings of Internet Measurement Conference, San Francisco, USA, pp. 267-271, November 2001.

[36] V. Jacobson, "Congestion Avoidance and Control", in Proceedings of IEEE ACM/SIGCOMM, pp. 314-329, August 1988.

[37] S. S. Kunniyur and R. Srikant, "An Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management," IEEE/ACM Transactions on Networking, vol. 12, No. 2, pp. 286-299, April 2004.

[38] J. F. Kurose and K. W. Ross, "Computer Networking: a top-down approach," Pearson Education, 2007

[39] T. V. Lakshman and U. Madhow, “The Performance of TCP/IP for Networks with high bandwidth-delay products and random loss,” IEEE/ACM Transactions on Networking, vol. 5, No. 3, pp. 336-350, June 1997.

[40] A. Lakshmikantha, R. Srikant, and C. Beck, “Impact of File Arrivals and Departures on Buffer Sizing in Core Routers,” in Proceedings of IEEE INFOCOM, Phoenix, USA, pp. 529-537, April 2008.

[41] L. Le, J. Aikat, K. Jeffay, and F. D. Smith, “The Effects of Active Queue Management and Explicit Congestion Notification on Web Performance,” IEEE/ACM Transactions on Networking, vol. 15, No. 6, pp. 1217-1230, December 2007.

[42] S. Leinen, “Use of RED in practice?” [Online]. Available: http://mailman.postel.org/pipermail/end2end-interest/2006-March/005851.html

[43] S. Liu, T. Basar and R. Srikant, “Exponential-RED: A Stabilizing AQM Scheme for Low- and High-Speed TCP Protocols,” IEEE/ACM Transactions on Networking, vol.13, No. 5, pp. 1068-1081, October 2005.

[44] S. H. Low, F. Paganini, J. Wang, and J. C. Doyle, “Linear Stability of TCP/RED and a Scalable Control,” Computer Networks, vol. 43, No. 5, pp. 633-647, December 2003.

[45] H. S. Martin, A. McGregor, and J. G. Cleary, “Analysis of Internet Delay Times,” in Proceedings of Passive and Active Measurement Workshop, Auckland, New-Zealand, April 2000.

[46] Vishal Misra, Wei-Bo Gong, and Don Towsley, “Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED,” in Proceedings of IEEE ACM/SIGCOMM, Stockholm, Sweden, pp. 151-160, 28 August-1 September 2000.

[47] M. Mathis, J. Semke, and J. Mahdavi, “The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm,” Computer Communication Review, vol. 27, No. 3, pp. 67-82, July 1997.

[48] M. May, J. Bolot, C. Diot and B. Lyles, “Reasons not to deploy RED,” in Proceedings of 7th International Workshop on Quality of Service, London, UK, 31 May-4 June 1999.

[49] T. J. Ott, T. V. Lakshman and L. Wong, “SRED: Stabilized RED,” in Proceedings of IEEE INFOCOM, New York, USA, pp. 1346-1355, March 1999.

[50] V. Paxson, “End-to-End Internet Packet Dynamics,” IEEE/ACM Transactions on Networking, vol. 7, No. 3, pp. 277-292, June 1999.

[51] S. Shenker and L.X. Zhang, “Some Observations on the Dynamics of a Congestion Control Algorithm,” Computer Communication Review, vol. 20, No. 5, pp. 30-39, October 1990.

[52] H. Sirisena, A. Haider, and K. Pawlikowski. “Auto-Tuning RED for Accurate Queue Control,” in Proceedings of IEEE GLOBECOM, Taipei, Taiwan, pp. 2010-2015, November 2002.

[53] J. Sun, K. Ko, G. Chen, S. Chan, and M. Zukerman, “PD-RED: To Improve the Performance of RED,” IEEE Communications Letters, vol. 7, No. 8, pp. 406-408, August 2003.

[54] L. Tan, W. Zhang, G. Peng, and G. Chen, “Stability of TCP/RED Systems in AQM Routers,” IEEE Transactions on Automatic Control, vol. 51, No. 8, pp 1393-1398, August 2006.

[55] K. Thompson, G. Miller, and R. Wilder, "Wide Area Internet Traffic Patterns and Characteristics," IEEE Network, vol. 11, No. 6, pp. 10-23, November-December 1997.

[56] C. Villamizar and C. Song, “High Performance TCP in ANSNET,” Computer Communications Review, vol. 24, No. 5, pp 45-60, October 1994.

[57] C. Wang, B. Li, Y. T. Hou, K. Sohraby, and Y. Lin, “LRED: A Robust Active Queue Management Scheme Based on Packet Loss Ratio,” IEEE Transactions on Parallel and Distributed Systems, vol. 18, No. 1, pp. 29-42, January 2007.

[58] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, “FAST TCP: Motivation, Architecture, Algorithms, Performance,” IEEE/ACM Transactions on Networking, vol. 14, No. 6, pp. 1246-1259, December 2006.

[59] H. Y. Zadeh, A. Habibi, H. Jafarkhani, and C. Bauer, “Optimal Statistical Tuning of the RED parameters,” in Proceedings of IEEE ICC, Beijing, China, pp. 27-32, May 2008.

[60] H.G. Zhang, C.V. Hollot, D. Towsley, and V. Misra, “A Self-Tuning Structure for Adaptation in TCP/AQM Networks,” in Proceedings of IEEE GLOBECOM, San Francisco, USA, pp. 3641-3646, December 2003.

[61] W. Zhang, L. Tan, and G. Peng, “Dynamic queue level control of TCP/RED systems in AQM routers,” Computers and Electrical Engineering, vol. 35, NO. 1, pp. 59-70, January 2009.

[62] Y. Zhang and L. Qiu, “Understanding the End-to-End Performance Impact of RED in a Heterogeneous Environment,” Cornell CS Technical Report 2000-1802, July 2000.

[63] T. Ziegler, S. Fdida, and C. Brandauer, “Stability Criteria for RED with Bulk-data TCP Traffic,” Technical Report, Aug. 1999, [Online]. Available: www-rp.lip6.fr/site_npa/site_rp/_publications/125-red_latest.ps.gz.

[64] T. Ziegler, S. Fdida, and C. Brandauer, “Stability Criteria for RED with TCP Traffic,” Technical Report, May 2000, [Online]. Available: http://www-rp.lip6.fr/~sf/WebSF/PapersWeb/red.net2000.pdf

[65] T. Ziegler, S. Fdida, C. Brandauer, and B. Hechenleitner, “Stability of RED with Two-way TCP Traffic,” October 2000, [Online]. Available: www-rp.lip6.fr/~sf/WebSF/PapersWeb/icccn.ps.

[66] T. Ziegler, C. Brandauer, and S. Fdida, “A quantitative Model for the Parameter setting of RED with TCP Traffic,” In Proc. of the Ninth International Workshop on Quality of Service (IWQoS), June 2001.

[67] “The Network Simulator - ns-2,” [Online] Available: www.isi.edu/nsnam/ns/

[68] Cisco System Inc., “NetFlow white papers,” [Online]. Available: http://www.cisco.com/en/US/products/ps6601/prod_white_papers_list.html

[69] Cisco System Inc., “Distributed Weighted Random Early Detection,” [Online] Available: http://www.cisco.com/en/US/docs/ios/11_1/feature/guide/WRED.html