UNIVERSITÉ DU QUÉBEC À CHICOUTIMI

AN ADAPTIVE ACTIVE QUEUE MANAGEMENT ALGORITHM IN INTERNET

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE EXTENSIONNÉE DE

L'UNIVERSITÉ DU QUÉBEC À MONTRÉAL

PAR

WANG JIANG

JUIN 2006

ABSTRACT

Random Early Detection (RED) algorithm a recommended active queue management scheme, that is expected to provide several Internet performance advantages such as minimizing packet loss and router queueing delay, avoiding global synchronization of sources, guaranteeing high link utilization and fairness. But this thesis shows that when the author experiments with RED by introducing services differentiation and investigates how the arrival of high priority UDP traffic can hurt the performance of lower priority TCP traffic when they share the same bottleneck link with one or two classes of service, RED does not minimize the number of dropped packets as expected. moreover it cannot make acceptable packet-dropping decision, especially, under heavy network load and high delay to provide high throughput and low packet loss rate. Thus the author introduces a new active queue management (AQM) algorithm Based on negative feedback control mechanism. AQM can be simulated by using the network simulator software NS-2. The simulation results show that AAQM (Adaptive Active Queue Management) provides postponing congestion and high link utilization whilst maintaining packet loss.

Keyword: AAQM RED TCP Negative Feedback Control NS-2

ACKNOWLEDGEMENT

First of all, I would like to express my deep and sincere gratitude to my supervisor, Professor Fan Xuemei, Ph.D. Her wide knowledge and her logical way of thinking have been of great value for me. Her understanding, encouraging and personal guidance have provided a good basis for the present thesis. What is more, her sense of humor towards life and enthusiasm for scientific research in general have been more than I could have ever asked for.

I wish to express my warm and sincere thanks to Yuan Wei, a classmate of mine, who helped me with the format of this thesis. Her valuable advice and friendly help have been very helpful for this study.

During this work I have discussed with many of my teachers and classmates for whom I have high regard, and I wish to extend my warmest thanks to all those who have helped me with my work in School of Tianjin University of Technology.

I owe my loving thanks to my father Wang Huili and my mother Jiang Fen, and especially to my beautiful and virtuous wife, Wu Xuemei, to whom this thesis is dedicated, who trusted and motivated me during my student career. Without their encouragement and understanding it would have been impossible for me to finish this work.

TABLE OF CONTENTS

Abstract ..... **ii**

Acknowledgement ..... **ii**

Table of Contents ..... **iii**

List of Figures ..... **vi**

List of Tables ..... **viii**

Chapter 1 Introduction **1**

Chapter 2 Background **4**

2.1 Transmission Control Protocol (TCP) 4

2.2 Protocol Operation 6

2.2.1 TCP Segment Structure 6

2.2.2 Connection Establishment 9

2.2.3 Data Transfer and Error Recovery 9

2.2.4 Connection Termination 12

2.3 Congestion Control 13

2.3.1 Slow Start 15

2.3.2 Congestion Avoidance 16

2.2.3 Fast Retransmit/Fast Recovery 17

Chapter 3 Router Queue Management 19

3.1 Passive Queue Management，PQM 19

3.1.1 Drop tail 19

3.2 Active Queue Management(AQM): 22

3.2.1 Random Early Detection(RED) 26

3.2.2 Explicit Congestion Notification ECN 30

3.3.3 BLUE 34

3.3.4 Stochastic Fair BLUE，SFB 36

3.3.5 Stabilized RED（SRED） 37

3.3.6 GRED 38

3.3.7 DRED (Dynamic RED) 39

3.3.8 SRED (Simple SRED) 40

3.3.9 Flow RED 42

3.3.10 Adaptive RED 43

3.3.11 RED with Penalty Box 44

3.3.12 GREEN 45

3.3.13 CHOKe（CHOose and Keep for responsive flows CHOose and Keep for unresponsive flows） 46

3.3.14 RIO（RED with In and Out） 46

3.3.15 Weighted RED 47

Chapter 4 AAQM Design 51

4.1 Base theory 51

4.2 Example 52

4.3 AAQM 54

Chapter 5 Performance evaluation and Simulation Results 59

5.1 Trial 1 61

5.2 Trial 2 62

5.3 Trial 3 64

5.4 Trial 4 65

5.5 Trial 5 66

Chapter 6 Conclusion and Future work 69

Bibliography 72

LIST OF FIGURES

figure 2-1 TCP segment format.....7

figure2-2 TCP three-way handshake.....10

figure 2-3 TCP connection tear-down.....13

figure 2-4 TCP *cwnd* variation during the phases of Slow Start and Congestion Avoidance.....17

figure 3-1 Average Queue VS Dropping P.....27

figure 3-2 RED algorithm pseudocode.....30

figure 3-3 ECN field in the IPv4.....31

figure 3-4 CE codepiont.....31

figure 3-5 SFB.....36

figure 3-6 adaptive RED pseudocode.....44

figure 4-1 negative feedback control.....51

figure 4-2 simplified scheme of adrenal steroidogenesis shows abnomal seretion of hormones in congenital adrenal hyperplasis resulting from 21-hyroxylase deficiency.....52

figure 4-3 Regulation of cortisol secretion in normal subjects and in pateints with congenital adrenal hyperplasia.....53

figure 4-4 AAQM mechanism design 57

figure 4-5 AAQM flow design 58

figure 5-1 simulaton topology 61

figure 5-2 RED actual queue 62

figure 5-3 AAQM actual queue 62

figure 5-4 first max queue time VS bandwidth 63

figure 5-5 queue limit VS packet loss rate 64

figure 5-6 Goodput VS queue limit 65

figure 5-7 average queuing delay VS queue limit size 65

figure 5-8 number of TCP session vs average packet loss rate 66

figure 5-9 queue limit VS packet loss rate. 67

figure 5-10 Goodput VS queue limit 68

figure 5-11 queue limit size VS average queuing delay 68

LIST OF TABLES

TABLE 3-1 AQM ALGORITHM COMPARE 50

CHAPTER 1

INTRODUCTION

When the aggregate demand for a resource (e.g., link bandwidth) exceeds the available capacity of the resource, Internet congestion occurs. Resulting effects from such congestion include long delays in data delivery, wasted resources due to lost or dropped packets, and even possible congestion collapse[1], in which all communication in the entire network ceases.

In order to maintain good network performance, certain mechanisms must be provided to prevent the network from being congested for any significant period of time. Two approaches to handle congestion are congestion control (or recovery) and congestion avoidance. [2] The former is reactive in that congestion control typically comes into play after the network is overloaded, i.e., congestion is detected. The latter is proactive in that congestion avoidance comes into play before the network becomes overloaded, i.e., when congestion is expected. Both approaches are used to denote the term congestion control.

Proposed one mechanism in congestion avoidance strategy[3], will have two components: The network must be able to signal the transport endpoints that congestion is occurring (or about to occur). And the endpoints must have a policy that decreases utilization if this signal is received and increases utilization if the signal is not received. If

packet loss is (almost) always due to congestion and if a timeout is (almost) always due to a lost packet, we have a good candidate for the 'network is congested' signal. Particularly since this signal is delivered automatically by all existing networks, without special modification (as opposed to [4] which requires a new bit in the packet headers and a modification to *all* existing gateways to set this bit). The most popular congestion avoidance algorithms in today's Internet are based on the window mechanism of the *Transmission Control Protocol (TCP).*

Congestion control includes:□The design of mechanisms and algorithms to statistically limit the demand-capacity mismatch. □Dynamically control traffic sources when such a mismatch occurs[5]. It has been shown that static solutions such as allocating more buffers, providing faster links or faster processors are not effective for congestion control purposes. Current usage of the Internet is dominated by transmission control protocol (TCP) traffic such as remote terminal (e.g., Telnet), FTP, Web traffic, and electronic mail (e.g., SMTP)[6].

However, current Internet congestion control methods are expected to result in unsatisfactory performance, e.g., multiple packet losses and low link utilization, as the number of users and the size of the network increase. Accordingly, many congestion control approaches have been proposed. Network algorithms such as active queue management (AQM), executed by network components such as routers, detect network congestion, packet losses, or incipient congestion, and inform traffic sources (either implicitly or explicitly). In response, source algorithms adjust the source's data-sending

rate into the network. The basic design issues are what to feedback (network algorithms) and how to react (source algorithms). Lefelhocz *et al.* [7,8] claim that packet scheduling, buffer management, feedback, and source algorithms (i.e., end-system adjustment) are four necessary and sufficient components for providing better best-effort services. They proposed a general design principle: the network should manage and distribute its resources through packet scheduling and buffer management and give the best possible explicit feedback[9]. In response, the source algorithms should implement the adjustments accordingly. Active queue management (AQM) was briefly introduced as a solution approach for congestion avoidance with a focus on the random early detection (RED) algorithm.

In this paper, I am trying to introduce a new AQM algorithm called AAQM that achieves this objective. The paper is organized as follows: in Chapter 2 we describe TCP .In Chapter3 current AQMs, pointing out their advantages and drawbacks. In Chapter 4 I am trying to introduce AAQM design. In Chapter 5 I am trying to demonstrate the performance of AAQM.And in last Chapter I will discuss conclusions and future works.

CHAPTER 2

BACKGROUND

The Internet Protocol (IP) provides an unreliable, connectionless datagram delivery service. IP does not guarantee to deliver correctly an IP datagram at its destination, it just assures a best effort service. The service is based on a connection less design, which means that the delivery of each datagram is treated independently of the other datagrams belonging to the same flow. This simple structure makes IP a very flexible and robust protocol, but the upper layers, such as TCP, have to provide a more reliable service and be able to recover from situations such as packet loss, packets out-of-order, damaged packets or duplicated packets [11].

2.1 Transmission Control Protocol (TCP)

The Transmission Control Protocol (TCP) guarantees reliable transportation of data and delivery of packets in order and without errors. Moreover it provides congestion control and a fair allocation of network resources. The most important aspects related to the TCP mechanism can be summarized in this way:

Connection-oriented

TCP is connection-oriented because before one application process can begin to send data to another, the two processes must first "handshake" with each other -- that is, they must send some preliminary segments to each other to establish the parameters of the ensuing data transfer. As part of the TCP connection establishment, both sides of the connection will initialize many TCP "state variables".

- Data Transfer

During the data transfer phase, a number of key mechanisms determine TCP's reliability and robustness. These include using sequence numbers for ordering received TCP segments.

- Error Control

Detecting duplicate data, checksums for segment error detection,and acknowledgements and timers for detecting and adjusting to loss or delay.

- Flow control

TCP offers efficient flow control, which means that, when sending acknowledgments back to the source, the receiving TCP process indicates the highest sequence number it can receive without overflowing its internal buffers.

- Multiplexing

TCP's multiplexing means that numerous simultaneous upper-layer conversations can be multiplexed over a single connection.

- Congestion control

Exponential opening of window back to 1/2 of where you were before congestion

loss.

2.2 Protocol Operation

Before proceeding we give an overview of the TCP segment format, the setting up of a connection, the actual data transfer and the connection closing.

2.2.1 TCP Segment Structure

All TCP data units that are used to set up a connection, transfer data and tear-down a connection have a standard format, shown in Figure 2.1. All segments are transferred between two TCP entities in the user data field of IP datagrams. The TCP segment format is composed by the header and by the data field of variable length. The normal size of the header is 20 bytes, unless other options are present[12].

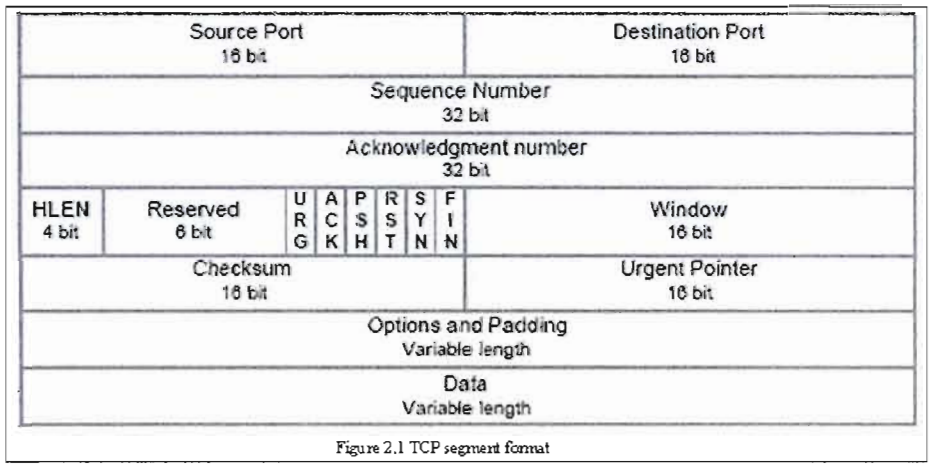


Figure 2.1 TCP segment format

figure 2-1 TCP segment format

The following fields characterize the TCP header:

- *Source Port* and *Destination Port*: the addresses of the end-points of the logical connection between two application protocols.
- *Sequence Number*: is used to differentiate segments and indicates the first byte in the data field of the segments relative to the start of the complete message. The number is linear so that the first segment sent has a number N and all subsequent segments have sequence numbers that relate to the number of bytes in the user data. For example, using a user data size of 1500 bytes and an initial sequence number 0, the first segment will have sequence number 0 and the next one 1500, the third 3000, etc.
- *Acknowledge Number*: the sequence number of the next byte this end of the connection is waiting for. All earlier bytes have been received successfully.
- HLEN: the total length of the TCP header. It indicates the number of 32-bit words it contains.
- *Reserved:* field reserved for future use.
- All TCP segments have the same header format and the validity of selected fields in the segment header is indicated by the setting of bits in the 6-bit *code-field*; if a bit is set (=1) the corresponding field is valid. Multiple bits can be set in a single segment. The 6 flags have the following meaning: URG is used if there are urgent data and in this case the *Urgent pointer* points to the urgent data; ACK contains a

valid number if it is an ACK packet; PSH is set to 1 when the sender wants to use the PUSH command; RST is used to reset the connection without a precise teardown; SYN is the synchronize flag essential for the connection set-up and FIN is used in the connection tear-down. Their use is explained later in this section.

- *Window*: specifies the Receiver Advertised Window (*rwnd*), which means the number of data bytes that the receiver is prepared to accept beyond the sequence number indicated in the acknowledgment number field (The maximum value is 65.535 bytes).
- *Checksum*: CRC calculated adding to the TCP header the IP address of the source and of the destination.
- *Options and Padding*: used to fill the header and to specify some options such as the MSS value in SYN segment (the default is 536 bytes till a maximum of 65535 bytes), the window's scale that defines the measure unit for the window length specified in *Window* (the default is 1byte) and the timestamp value, explained later.

2.2.2 **Connection Establishment**

TCP is oriented to the transmission of a continuous stream of octets and converts the data flow in segments suitable for transmission through IP. The application passes the data to a TCP buffer and TCP builds segments out of them and transmits them. The segment size is limited by the Maximum Segment Size (MSS). TCP is a

connection-oriented protocol: before end-nodes can exchange data, it is necessary to establish a connection between them[13]. In order to avoid any ambiguity with the initial sequence number setting at both sides of a connection, each side informs the other of the initial sequence number it proposes to use.

The connection establishment is realized through the three-way handshake mechanism, as shown in Figure 2.2. In this phase the two hosts negotiate the MSS of the connection. The requesting end sends an initial sequence number to the receiver using a SYN chronized packet (flag SYN=1). The remote host first stores the sequence number setting for the incoming direction, then it answers with an SYN/ACK message to acknowledge the sender's initial sequence number and to communicate its initial sequence number. Finally the initiating side responds with an ACK to the remote host's sequence number. The connection is now established.

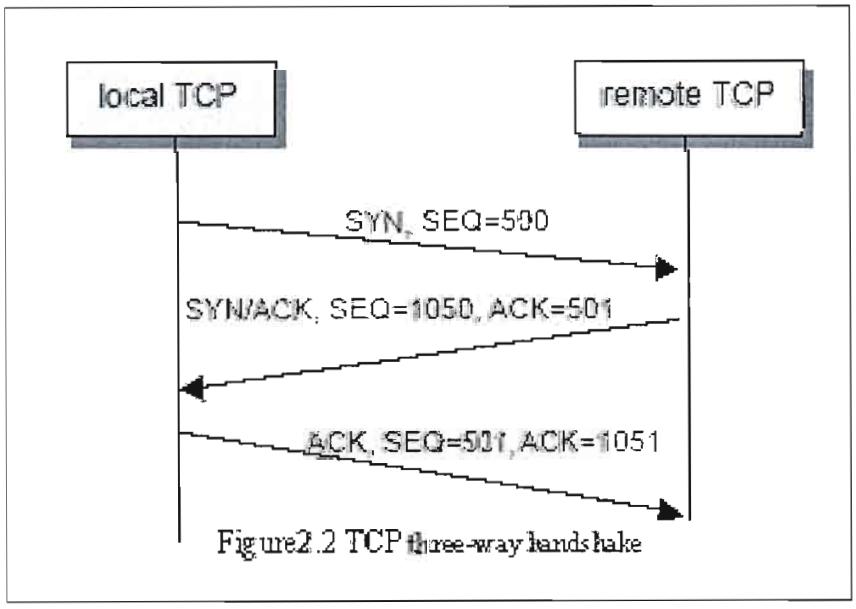


figure2-2 TCP three-way handshake

2.2.3 Data Transfer and Error Recovery

The error and flow control functions are the main procedures associated with the data transfer. They are employed with the intent to ensure that all the segments are successfully sent and acknowledged by the receiver and finally guarantee the reliable delivery of data.

TCP implements error control functions based on the go-back mechanism of retransmission. If the data have not been received within the timeout the segment has to be retransmitted. TCP assigns an exclusive sequence number for each octet transmitted and in the TCP header of each segment there is the number of the first octet of that segment. The receiver sends an acknowledgment (ACK) upon reception of a segment. The acknowledgments are cumulative: an ACK confirms all the bytes up to the given sequence number. In the ACK the sequence number of the next expected octet is carried. The sequence number is used to reconstruct the order of the segments received. Errors can be discovered thanks to the checksum.

TCP has a significant property of self-clocking, in the equilibrium state, each arriving ACK triggers a transmission of a new segment. Generally, TCP does not acknowledge a received segment immediately, but waits for a certain time in order to reduce the traffic on the link. In fact, if a data segment is sent during this time, the acknowledgment is piggy-backed into it [14]. If there are no packets out-of-order, no errors, no duplicate packets, all the data are acknowledged, buffered at the receiver and then

delivered to the upper layer. Otherwise, if the TCP host receives duplicate packets or out-of-order segments it does not acknowledge new data, but immediately sends an ACK segment that acknowledges the highest sequence number correctly received so far. This means that in these cases the sender may receive *duplicate acknowledgments* (DUPACK), which acknowledges the same segment as the previous ACK. When the first DUPACK is received TCP waits to retransmit the packets not yet acknowledged and when the three DUPACKs are received the Fast Retransmit algorithm is triggered. We will explain it later. In combination with the retransmission timer (RTO), on the sender side, ACKs provide reliable data delivery. A lost packet is generally indicated by the expiration of the RTO or the receipt of a duplicate acknowledgment.

To prevent a fast sender from overflowing a slow receiver, TCP uses the *Flow Control,* based on the principle of the *sliding window* mechanism. The receiver specifies the dimension of the *Receiver* Advertised Window (rwnd) in each data it sent to the opposite end, as an indication of the amount of data it is able to receive. An arriving ACK allows more data to be transmitted by advancing the sliding window to the right. When the total size of outstanding segments, segments in flight (FlightSize), fills up the receiver advertised window, the transmission of data is stopped until the sliding window advances or a larger receiver window is advertised. Specifying a *rwnd* of zero bytes is possible and can be used to force the sender into the persist mode. In this state the connection is still alive, but the transmission of new data is blocked.

Finally, to achieve Multiplexing communication, TCP allows the use of multiple ports

within a single host. A socket is defined as the couple port or address host and network address. A pair of sockets identifies unambiguously every connection, thus a single socket can be active simultaneously in several connections.

2.2.4 Connection Termination

When the communication is complete the connection is closed and the resources are released. The connection tear-down is shown in Figure 2.3. When one side of the system wants to terminate the connection, it sends a FIN segment (flag FIN=1) to the other side. As a reply the node on the other end returns an ACK segment to acknowledge the receipt of the FIN segment. The connection is still open for the other direction and the host on the other side can go on sending data. When it has finished it sends a FIN message with the last data and waits for the acknowledgment. When it receives the last acknowledgment, it finally closes the connection. If segments disappear in the middle of the connection termination, they are resent with the usual retransmissions.

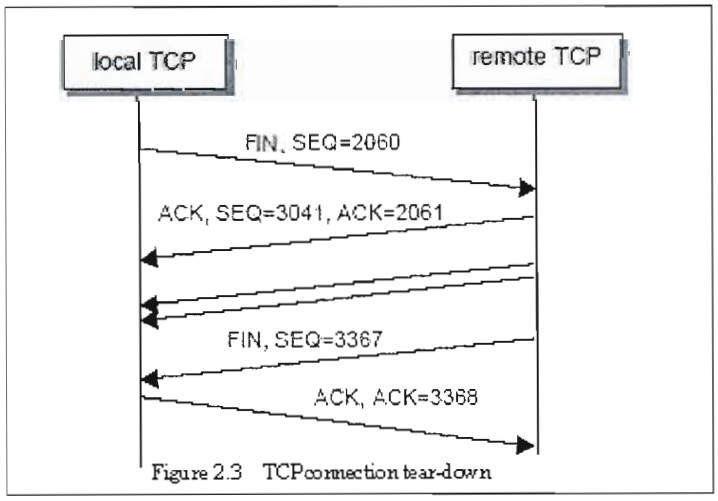


figure 2-3 TCP connection tear-down

2.3 Congestion Control

When one or several TCP connections are sending at inappropriately high rates the network can suffer from congestion. The router buffers saturate and some packets or their corresponding ACKs may be dropped before reaching their destination. This occurs when routers are receiving more packets than they can handle. Congestion collapse is a state in which packets are being injected into the network, but very little useful work is being accomplished. The specification of the *Receiver* Advertised Window (rwnd) is a way to control the rate of data incoming at the receiver, but is not enough to prevent network congestion.

Early in its evolution, TCP was enhanced by the congestion control mechanism to protect the network against the incoming traffic that exceeds its capacity. The first job of a congestion avoidance mechanism at the gateway is to detect incipient congestion and as stated in [15] a congestion avoidance scheme maintains the network in a region of low delay and high throughput. The congestion control is implemented in the hosts and consequently is an end-to-end congestion control. When a network is congested, more connections compete for limited resources. The congestion control mechanism tries to regulate the transmission rate of each connection in order to provide a fair sharing of network resources.

TCP congestion control is window-based. The sender rate is regulated by a Congestion Window (cwnd) that limits the amount of data a sender can have outstanding

in the network. It must not send data with a sequence number higher than the sum of the last acknowledged packet and the minimum of *cwnd* and *rwnd*. The cwnd changes its value in relation to the events the sender observes. The basic principle on which the TCP congestion control is built is to consider the packet loss as a signal of congestion. Thus the reaction to this event is to reduce the cwnd.

Two algorithms, known as Slow Start and Congestion Avoidance, regulate the reduction and the increasing of cwnd [15], basing their evaluation on a further variable: the slow start threshold (*ssthresh)*. Depending on its value the sender is subject to a Slow Start (cwnd<SSTHRESH< SPAN>) or Congestion Avoidance (cwnd>ssthresh) mechanism.

2.3.1 Slow Start

The Slow Start algorithm is based on the observation that the rate at which packets are injected into the network should be regulated by the rate at which acknowledgements arrive from the receiver should be the same. It is used at the beginning of the connection to avoid congesting the network or after repairing a packet loss detected by time-out expiration. The initial value of cwnd must be no more than 2 segments and the *ssthresh* may be arbitrarily high. Being *cwnd*<>the Slow Start algorithm is used. During this phase *cwnd* is increased by at most *sender maximum segment size* (SMSS) bytes for each ACK received that acknowledges a new packet. The sender can transmit up to the minimum of the *cwnd* and *rwnd* as said before. The growth of *cwnd* is exponential: cwnd

roughly doubles per each RTT. Slow Start ends when *cwnd ‡ ssthresh* or when congestion occurs. Congestion is declared when a timeout expires or three consecutive DUPACKs are received. They are both indications of a packet loss.

If a timeout expires TCP reacts decreasing *ssthresh* to be half of the number of segments outstanding into the network (*flightsize*):

$$ssthresh=\max\left(2,\min\left(\frac{flightsize}{2},rwnd\right)\right)$$

The retransmission timer is re-calculated through the exponential backoff (the new RTO is equal to the old one multiplied by a constant value, greater than 2) and cwnd=1. Now cwnd< *ssthresh* so a new phase of Slow Start is entered. But now *ssthresh* is lower than before and this implies that the capacity of the network will not saturate quickly as previously. The sender starts to retransmit packets beginning from which has caused the timeout expire.

2.3.2 Congestion Avoidance

The second situation to analyse is when Slow Start ends because cwnd=ssthresh (or cwnd>ssthresh). *In this case Congestion Avoidance sta rts. During this phase cwnd is* incremented by 1 SMSS only after a full window of data is acknowledged:

$$cwnd = cwnd + SMSS * SMSS/cwnd$$

for every incoming new ACK. This means that if *cwnd* permits to transmit N packets only after the reception of N ACK relative to all the packets sent, *cwnd* will increase by 1

packet (SMSS bytes). The growth of *cwnd* is linear. Figure2.4 illustrates an example of *cwnd* variation during the phases of Slow Start and Congestion Avoidance.

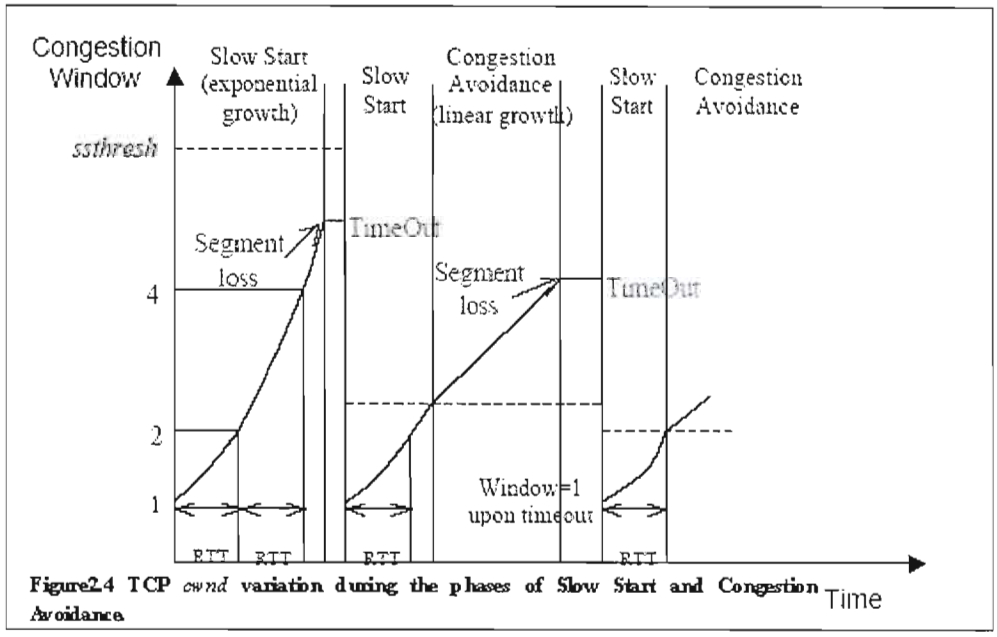


Figure2.4 TCP *cwnd* variation during the phases of Slow Start and Congestion Avoidance

figure 2-4 TCP *cwnd* variation during the phases of Slow Start and Congestion Avoidance

2.2.3 **Fast Retransmit/Fast Recovery**

The last case to analyse is what happens when three DUPACKs are received and congestion is declared. In this description we refer to the so-called Reno algorithm described in RFC 2581 [16]. It also introduces the use of TCP Selective Acknowledgment [17].

A TCP receiver should send DUPACK as soon as possible when an out-of-order packet arrives. The meanings of a DUPACK for a sender can be various: a packet drop, a

reordering of data segments and duplication of ACKs or data segments. So when the first DUPACK is received TCP waits to retransmit the supposed packet drop because the cause may be different from that of a packet discarded. The fast retransmit algorithm uses three DUPACKs as indication of packet loss. The Fast Retransmit and Fast Recovery behave according to this model.

1. When the third DUPACK arrives *ssthresh* changes:

$$ssthresh = \max\left(\frac{flightsize}{2}, 2 * SMSS\right)$$

2.The lost packet is retransmitted.

3.*cwnd=ssthresh + 3*SMSS,* this inflates the congestion window by the number of packets (three) that have left the network and are in the receiver buffer.

4.Increase *cwnd* by one SMSS for each additional DUPACK received.

5.Transmit a packet if allowed by the new value of cwnd and rwnd.

6.When an ACK that acknowledges new packets arrives, set *cwnd* to the value *ssthresh* set in step 1. The Fast Retransmit phase ends.

The assumption at the basis of these two algorithms is that if DUPACKs arrive it means that a packet has been lost. But if DUPACKs arrive it also means that after the packet loss data segments have arrived at the receiver buffer and this allows increasing the *cwnd* towards its previous value on arrival of each DUPACK. The problem is, what happens when there are multiple packets dropped.

If the SACK (Selective Acknowledgment) option is not available the sender has

little information about which packets to retransmit during Fast Retransmit. When it receives the acknowledgment for the retransmitted packet in the case of multiple drops it receives a partial acknowledgment and this will acknowledge some but not all packets sent before the Fast Retransmit[18]. Therefore, the TCP sender has no other choice than wait for retransmission timeout to expire.

CHAPTER 3

ROUTER QUEUE MANAGEMENT

The current router queue management is divided into the two major kinds: Passive Queue Management(PQM) and Active Queue Management，(AQM).

3.1 Passive Queue Management ，PQM

3.1.1 Drop tail

Traditional Internet routers employ the *Tail Drop* discipline for managing the buffer queue occupancy. It simply sets a maximum length for each buffer queue and it enqueues packets until the maximum length is reached, then drops subsequent incoming packets until the queue is decreased below its maximum value. Such a mechanism allows the router to maintain high queue occupancy, which is clearly undesirable since it tends to discriminate against bursty traffic and to drop many packets at the same time producing global synchronization of sources. Drop-Tail are intended to buffer bursty packet arrivals before forwarding on the outbound interface. Arriving packets are enqueued at the tail of the queue. When the queue fills, the arriving packet is discarded. The "drop-tail" behavior is effective in providing congestion notification to responsive flows as demonstrated by

the success of the Internet. However, with drop-tail, the decision to drop a packet is essentially a passive one. Moreover, drop-tail has several flaws that prompted research into a more active approach to router queue management . These flaws are most apparent during periods of persistent congestion and include the problems of *lock-out* and *full queues*. Briefly, *lock-out* refers to a phenomenon where a few flows are able to monopolize the queue space. Because of TCP synchronization effects, packets from some flows always arrive to a full-queue and are subsequently dropped, effectively locking those flows out of the outbound link and preventing them from making progress. (For more details, see.) *Full queues* also occur during persistent congestion. When the queue is consistently full, it is simply a source of latency and cannot serve its intended function of buffering bursty arrivals. Moreover, because drop-tail routers only drop packets when the queue is full, sources are only able to detect and respond to congestion after it has grown persistent. Notification of imminent congestion would allow sources to activate their avoidance mechanisms before congestion becomes severe.

Being the data traffic in Internet inherently bursty, one way to alleviate the problem is to provide the routers with fairly large buffers in order to absorb burst arrivals of packets to reduce losses and hence maintain high link utilization. On the other hand, large buffers tend to increase queueing delays at congested routers. The traditional Tail-Drop buffer management forces network applications to choose between high utilization or low delay.

The common method for managing router queue lengths is to fix a maximum length

(in terms of packets) for each queue, accept packets for the queue until this limit is reached, then refuse subsequent incoming packets until the queue decreases because a packet from the queue has been transmitted[19]. This technique is known as Tail Drop, since the packet that arrived most recently, the one on the tail of the queue, is rejected and discarded when the queue saturates. This is the traditional mechanism that has been used in the Internet for years, but it has five important disadvantages, as stated in.

①When network congestion occurs ,it only simply treat congestion and can not avoid congestion occurrence.

② For the Drop Tail algorithm, each congestion period will cause the global synchronization phenomenon in the network :when the queue overflows, the packet in the queue will be throw away in the same time .Those connection simultaneously reduce their sending out window, causing network traffic loss;

③ Do not have priority concept ;

④ Do not distinguish UDP flows and TCP flows, TCP flows is disadvantageous in resources competing, not having fair guarantee.

⑤Lock-Out The tail drop mechanism may allow a monopolization of queue resources by a single or few flows, denying other connections the possibility to find place in the router buffer.

⑥Full Queues Since congestion is detected only after a packet has been discarded and this occurs just when the queue router is full, Tail Drop allows queues to reach a full status and to persist in this steady state for long. Instead, the average queue size should be

kept low and fluctuations in the actual queue size should be allowed, thus to accommodate bursty traffic

Approaches like Random drop on full or Drop front on full are similar to Tail Drop and do not solve the problem of full queues. The only difference they have from Tail Drop is the criteria used to choose which packet to discard when the queue saturates. The former chooses randomly a packet in the queue and the latter drops the packet at the front of the queue. In order to solve the problem of full queues in routers this basic queue-admission policy can be modified by establishing admission criteria that direct which packets may be discarded even though space is available on the queue, but near to congestion.

3.2 Active Queue Management(AQM):

It is useful to distinguish between two classes of router algorithms related to congestion control: "queue management" versus "scheduling" algorithms. To a rough approximation, queue management algorithms manage the length of packet queues by dropping packets when necessary or appropriate, while scheduling algorithms determine which packet to send next and are used primarily to manage the allocation of bandwidth among flows. While these two router mechanisms are closely related, they address rather different performance issues.

- THE NEED FOR ACTIVE QUEUE MANAGEMENT

The native assumption might be that there is a simple tradeoff between delay and

throughput, and that the recommendation that queues be maintained in a "non-full" state essentially translates to a recommendation that low end-to-end delay is more important than high throughput.

The point of buffering in the network is to absorb data bursts and to transmit them during the (hopefully) ensuing bursts of silence. This is essential to permit the transmission of bursty data. It should be clear why we would like to have normally-small queues in routers: we want to have queue capacity to absorb the bursts.

The solution to the full-queues problem is for routers to drop packets before a queue becomes full, so that end nodes can respond to congestion before buffers overflow. We call such a proactive approach "active queue management". By dropping packets before buffers overflow, active queue management allows routers to control when and how many packets to drop.

1. Reduce number of packets dropped in routers. By keeping the average queue size small, active queue management will provide greater capacity to absorb naturally-occurring bursts without dropping packets.
2. Provide lower-delay interactive service By keeping the average queue size small, queue management will reduce the delays seen by flows.
3. Avoid lock-out behavior Active queue management can prevent lock-out behavior by ensuring that there will almost always be a buffer available for an incoming packet. For the same reason, active queue management can prevent a router bias against low bandwidth but highly bursty flows. Active queue management is needed to control

the overall average queue sizes, so that arriving bursts can be accommodated without dropping packets. In addition, active queue management should be used to control the queue size for each individual flow or class, so that they do not experience unnecessarily high delays. Therefore, active queue management should be applied across the classes or flows as well as within each class or flow.

- THE FLOW CLASSIFICATIOM

It is convenient to divide flows into three classes: (1) TCP- compatible flows, (2) unresponsive flows, i.e., flows that do not slow down when congestion occurs, and (3) flows that are responsive but are not TCP-compatible. The last two classes contain more aggressive flows that pose significant threats to Internet performance, as we will now discuss.

The projected increase in more aggressive flows of both these classes, as a fraction of total Internet traffic, clearly poses a threat to the future Internet. There is an urgent need for measurements of current conditions and for further research into the various ways of managing such flows. There are many difficult issues in identifying and isolating unresponsive or non-TCP-compatible flows at an acceptable router overhead cost. Finally, there is little measurement or simulation evidence available about the rate at which these threats are likely to be realized, or about the expected benefit of router algorithms for managing such flows.

There is an issue about the appropriate granularity of a "flow". There are a few "natural" answers: 1. a TCP or UDP connection (source address/port, destination

address/port); 2. a source/destination host pair; 3. a given source host or a given destination host. We would guess that the source/destination host pair gives the most appropriate granularity in many circumstances. However, it is possible that different vendors/providers could set different granularities for defining a flow (as a way of "distinguishing" themselves from one another), or that different granularities could be chosen for different places in the network. It may be the case that the granularity is less important than the fact that we are dealing with more unresponsive flows at *some* granularity. The granularity of flows for congestion management is, at least in part, a policy question that needs to be addressed in the wider IETF community.

Internet routers should implement some active queue management mechanism to manage queue lengths, reduce end-to-end latency, reduce packet dropping, and avoid lock-out phenomena within the Internet [20]

The default mechanism for managing queue lengths to meet these goals in FIFO queues is Random Early Detection (RED). Unless a developer has reasons to provide another equivalent mechanism, we recommend that RED be used.

In order to solve the problem of full queues in routers this basic queue-admission policy can be modified by establishing admission criteria that direct which packets may be discarded even though space is available on the queue, but near to congestion. Such an algorithm does not eliminate packet drops, but on the contrary, it discards or marks packets at an earlier stage in order to solve the congestion problem for flows that are responsive to packet drops as congestion signal. It has four advantages, as stated in.

First of all it allows to reduce the number of packets dropped in the router, because keeping the average queue size small the router buffer can absorb bursts without discarding packets, or better, dropping less packets in the case of queue overflows than in the network implementing Tail Drop or other algorithms.

It solves the problem of synchronization: Active queue management schemes may avoid this synchronization giving the possibility to reduce their sending rate at different moments and not all the same moments. This permits to achieve higher throughput. It provides lower delay on the link thanks to the reduced size of the buffer queue. It solves the problem of lock-out, as defined before.

3.2.1 Random Early Detection(RED)

RED algorithm control average queue length in a low level, make router to keep higher throughout and the packet also get the small delay. Keeping actual average queue in a lower length, the actual queue can receive more data. With RED[21,22], a link maintains an exponentially weighted moving average (EWMA) queue length, $avg = (1 - w_Q) \times avg + w_Q \times Q$, where Q is the current queue length and w_Q is a weight parameter, $0 \leq w_Q \leq 1$. When avg is less than the minimum threshold (min_{th}), no packets are dropped (or marked). When it exceeds the maximum threshold (max_{th}), all incoming packets are dropped. When it is in between, a packet is dropped with a probability pa that is an increasing function of avg. More specifically, if $min_{th} \leq avg \leq max_{th}$, then the temporary dropping (or marking) probability, p_b, is calculated as:

$$p_b = \max_p * \frac{avg - \min_{th}}{\max_{th} - \min_{th}}$$

where max_p is the maximum value of p_b. Then p_a is calculated as:

$$p_a = \frac{p_b}{(1 - count * p_b)}$$

where *count* is the number of undropped packets since the last dropped packet. In this way RED drops (or marks) packets in proportion to the input rates of the connections. Connections with higher input rates receive proportionally more drops (or marks) of packets than connections with lower input rates. By doing so, RED tries to maintain equal rate allocation and remove biases against burst connections.

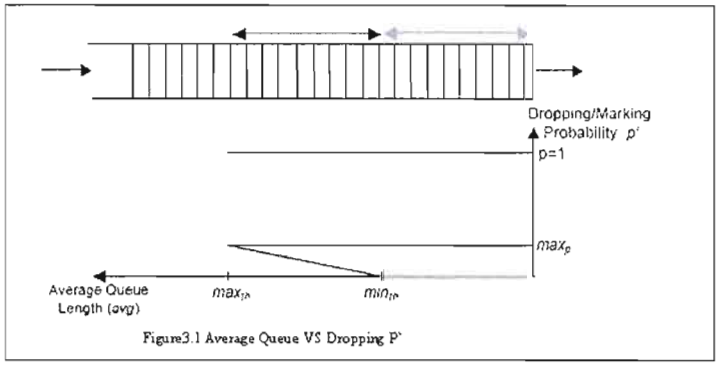


figure 3-1 Average Queue VS Dropping P

By using probabilistic packet dropping RED also eliminates global synchronization. The RED algorithm realizes following purpose[23]:

congestion avoid: RED router throw away packet in the earlier period before congestion occurrence , make source end enter congestion avoiding stage in order to avoid network congestion occurrence;

congestion control: The RED algorithm can control the max queue length in the router to

deal with the network congestion ,although the transport layer lack to cooperate;

Avoid the global synchronization phenomenon: Packet throw away packet by random fashion. Many connections increasing or reducing their sending out window at the same time is impossible, so we can avoid the global synchronous phenomenon;

the fair competition of each UDP flows: the drop packet number of An UDP flow should take proportion with be occupied bandwidth in RED algorithm.

minimize packet loss and queueing delay

maintain high link utilization and maximize the *global power* (the ratio of throughput to delay)

be useful for a wide range of environments, with a variable number of connections with different round trip times, data loads and throughput

But, RED algorithm have two problems [24]:

①Fairness problem: mostly express in two aspects: One is fairness of each TCP connection, some time packet discard rate is same to each connection, but connection with smaller RTT compete bandwidth in advantage, causing not fairness; Two is fairness of TCP flow with UDP flow, the TCP flows respond to packet discard ,but UDP flow not. the result makes UDP flows occupy more bandwidth.

② Have the priority problem: RED algorithm does not have priority concept, can't adapt to the commercial actuality in Internet.

Unlike Tail Drop where the only free parameter is the buffer size in RED we have several parameters to set, like *wq*, *minth* ,*maxth* and *maxp*. In order to achieve a good congestion

avoidance the parameter sensitivity must be kept low. Some rules to follow are suggested in [25,26] and shortly we summarize them:

1. set *minth* high enough to guarantee a high utilization of the link. The optimal value for it depends on the link speed, propagation delay and buffer size. In the ns-2 simulator, *minth* is set as default to 5 packets (or, for a queue measured in bytes rather than packets, 5 packets times the mean-packet-size in bytes) and according to the link characteristics it seems to be a good rule not to keep it too small, such as 1 or 2 packets because we do not allow enough burstiness in the arrival process.

2. *maxth-minth* must be kept large enough to avoid global synchronization. If it is small, *avg* can oscillate regularly from the minimum till the maximum threshold and the gateway will discard a lot of packets in the same moment. The [FJ93] paper recommends setting *maxth* to at least twice *minth*, but the current rule of thumb is to set *maxth* to three or more times *minth.* In the ns-2 simulator *minth* and *maxth* are set to 5 and 15 packets respectively.

3. set *wq* ‡ 0.001 and vary that according to the burst length L we want to be able to accommodate, without dropping any packets. For example if *minth* is 5 packets and L is 50 packets, then *wq* has to be around 0.0042. In the ns-1 and ns-2 simulator it is set to 0.002. That is why if *wq* is too low, then *avg* is responding too slowly to transient congestion. If it is too high, then the estimated average queue size is too dependent on the instantaneous queue size.

The conclusion is that unfortunately there are no precise optimal values because they depend on a wide range of factors, including not only the link speed and propagation delay, but also the traffic characteristics. RED algorithm follow as:

```
for each packet arrival
   calculate the average queue size avg
   if min_th ≤ avg < max_th
      calculate probability p_a
      with probability p_a:
         mark the arriving packet
   else if max_th ≤ avg
      mark the arriving packet
```

figure 3-2 RED algorithm pseudocode

3.2.2 Explicit Congestion Notification ECN

The Active queue management algorithm detects congestion before the buffer queue overflows and signals to the end nodes the incipient congestion. As mechanism for congestion indication the router may drop packets, but a more efficient way to achieve that is marking packets. The router may use the Congestion Experience (CE) codepoint in the IP packet header as an indication of congestion for the end-nodes.

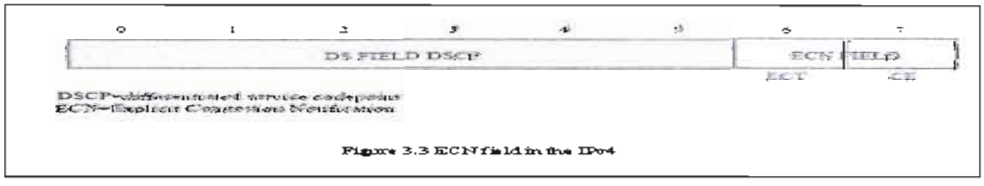
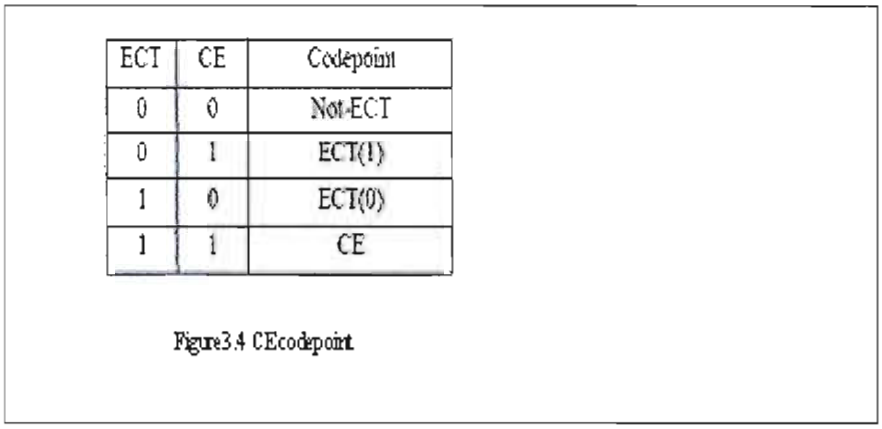


Figure 3.3 ECN field in the IPv4

figure 3-3 ECN field in the IPv4

Considering the IP header in the IPv4 TOS octet[25], the ECN field is defined through the bits 6 and 7, defined for experimental use of ECN. Marking packets is done through the ECN field and there are four possible ECN codepoints. In the ECN field we distinguish the ECT (ECN-Capable Transport) bit and the CE bit. The ECN-Capable Transport codepoints, ECT (0) and ECT (1), mean that the end-points are ECN-capable. The Not-ECT codepoints means that the packet is not using the ECN field. Then there is the CE codepoint used to signal the presence of congestion.



ECT	CE	Codepoint
0	0	Not-ECT
0	1	ECT(1)
1	0	ECT(0)
1	1	CE

Figure3.4 CEcodepoint

figure 3-4 CE codepiont

When the router is ready to discard packets to signal end-points of incipient congestion, at the reception of a new packet it has to check the ECT codepoint. If is set (ECN-capable flows) it should set the CE codepoint and forwards the packet, or on the contrary (not ECN-capable flows) discard packet. When the end-nodes receive a packet ECN-capable with the CE bit set, the reaction will be the same to the detection of a drop

packet. It is important that the router sets the CE codepoint only in the case it would have discarded the packet. If the router is not in the presence of incipient congestion it just has to forward the packet without modifying the CE codepoint. We expect that using ECN in combination with RED the router will set the CE codepoint when the average queue size exceeds the fixed threshold, instead of dropping the packet. It is evident that the measures taken by the router against congestion must not be based on the instantaneous value of the queue, but on the average queue size, because this is subject to frequent variations.

The use of two ECN-Capable Transport codepoints is due to several reasons and one of them is that it makes it more difficult for a router to erase the CE codepoint, without being discovered by the end nodes, since they have to be able to reconstruct the original one.

In order to support the use ECN field some modifications are requested to TCP level.

(1)First of all, during connection set-up, the two end nodes must negotiate if they are both ECN capable

(2) Then there is a need to find a way to communicate the reception of a CE packet from the receiver to the sender

(3) and to signal the corresponding reduction of the congestion window from the sender to the receiver .

(4)In order to support ECN functionality, two new flags are defined in the TCP header: the Explicit Congestion Notification (ECE) and the Congestion Window Reduced (CWR)

flags.

During the connection setup the two end-hosts decide whether to support the ECN functionality. They do that by exchanging SYN packets with ECE and CWR opportunely set[26]. If both the nodes are ECN-capable they will use ECN notification during the connection setting in the IP-header of each packet a ECT codepoints.

The receiver at the reception of a CE packet can inform the sender of the incipient congestion setting the ECN-Echo (ECE) flag in the TCP header. The sender seeing the ECE flag set infers that congestion was encountered in the network on the path from the sender to the receiver and reacts in the common way, as TCP congestion control indicates: it halves the congestion window and reduces the slow start threshold. It is important to notice that the TCP sender should react to an ECN at most once per roundtrip time. The TCP should ignore subsequent ECNs if the source has reacted to a previous ECN or to a dropped packet in the last round trip time.

When an ECN-capable TCP sender reduces the congestion window it will set the CWR flag in the TCP header of the first new packet sent to inform the data receiver that the congestion window has been reduced. The TCP receiver uses the CWR flag received to determine when to stop setting the ECN-Echo flag in the TCP header. Several simulations show that there are several advantages deriving from the introduction of the ECN mechanism in TCP networks. A main benefit of the ECN scheme is in avoiding unnecessary packet drops. This is of great importance for interactive traffic and for low-bandwidth traffic where the user is delay-sensitive. Moreover, ECN provides a fast

and clear mechanism to inform the sender of incipient congestion, sparing it to wait for a retransmit timeout or the reception of three DUPACKs.

Active queue management mechanisms may use one of several methods for indicating congestion to end-nodes. One is to use packet drops, as is currently done. However, active queue management allows the router to separate policies of queuing or dropping packets from the policies for indicating congestion. Thus, active queue management allows routers to use the Congestion Experienced (CE) codepoint in a packet header as an indication of congestion, instead of relying solely on packet drops. This has the potential of reducing the impact of loss on latency-sensitive flows[27,28,29].

On the other hand there are two potential disadvantages of ECN[30,31]. One is related to the case of non-compliant TCP connections that, even being ECN capable, ignore ECN notifications, but this is a problem also for networks that based congestion control on packet drops. The other drawback is specific for the ECN mechanism and occurs when an ECN message is discarded by the network and the notification of congestion cannot reach the end-node. But if the routers implement RED they will go on calculating the average queue size and mark packets randomly, as long as the congestion persists.

3.3.3 BLUE

BLUE differs from all the classical algorithms such as RED[32] and derivations from RED based on the control of the average queue variations, since it is based directly on

packet loss and link utilization history. BLUE uses a unique marking/dropping probability pm. If the router buffer saturates continually and packets are discarded, pm is incremented. On the other hand, if the queue is almost empty or the link idle the pm is reduced. This allows BLUE to learn the correct rate it needs to send back a congestion notification. BLUE is a hybrid flow and queue based congestion control scheme. It uses packet loss (queue) and link under-utilization (flow) events to adjust the rate of congestion notification. The congestion notification rate *pm* is increased at a set rate if the queue size exceeds a threshold L, and it is decreased if the link is idle. The amount of increase is *d1*, and decrease *d2*, at a rate of 1/freezetime.

There is a subtle drawback[33] with BLUE that may limit its practicability. BLUE behaves well if operated within a region of RTT and number of connections N for which the parameters d1 and d2 were set. However, changes in the dominant RTT of the connections going through the queue, or a significant change in N can invalidate the parameter settings and lead to queue backlog oscillation between loss and under-utilization. The amount of change of notification rate *pm* during a queue full or link idle event, is the product of the time spent in this event multiplied by the rate of change d(1,2)/freezetime. This time is related to the delay in the response of the TCP sources to the changed notification rate (2 x RTT). The greater the RTT, the greater will be the *pm* adjustment. If the RTT increases, so does the change in *pm* and this may result in backlog oscillation. This was observed in simulations using the recommended parameter settings of. Another cause of instability is a large change in the number of

connections. It is not our intention to explore BLUE in depth here, but this instability is the result of the adjustment of congestion notification rate *pm* by a constant d1 or d2, despite the non linear relation of *pm* and N. Recall that based on the TCP Friendly Equation the function of *pm* versus N requires larger changes of *pm* for larger N.

3.3.4 Stochastic Fair BLUE, SFB

SFB is highly scalable and enforces fairness using an extremely small amount of state and a small amount of buffer space. SFB is based on two independent algorithms[34]. The first is the BLUE queue management algorithm. This algorithm uses a single marking probability to mark packets (using ECN [14]) in times of congestion. The heavier the congestion is, the higher the marking probability. The second algorithm is based on Bloom filters. This algorithm allows for the unique classification of objects through the use of multiple, independent hash functions. Using Bloom filters, object classification can be done with an extremely small amount of state information.

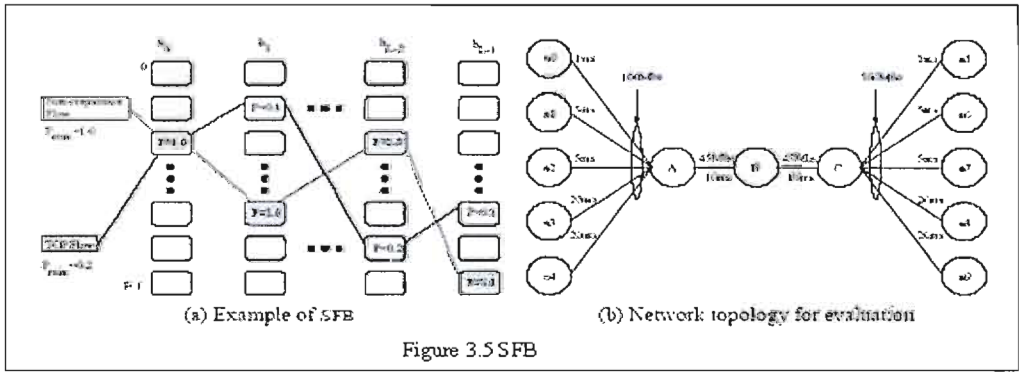


Figure 3.5 SFB

figure 3-5 SFB

SFB combines BLUE and Bloom filters to produce a highly scalable means to enforce fairness amongst flows using an extremely small amount of state and a small amount of buffer space. SFB is a FIFO queueing algorithm that identifies and rate-limits non-responsive flows based on accounting mechanisms similar to those used with BLUE. SFB maintains $N \times L$ accounting bins. The bins are organized in L levels with N bins in each level. SFB also maintains(L)independent hash functions, each associated with one level of the accounting bins. Each hash function maps a flow into one of the N accounting bins in that level. The accounting bins are used to keep track of queue occupancy statistics of packets belonging to a particular bin. This is in contrast to Stochastic Fair Queueing (SFQ) where the hash function maps flows into separate queues.

3.3.5 Stabilized RED（SRED）

Stabilized RED is a RED-derived mechanism that attempts to improve RED performance by considering a further element in calculating discard or marking probability, that is, the estimated number of active connections or flows. The basic idea for estimating the number of active flows is to compare, every time a packet arrives at the buffer, the new packet with one entry randomly taken from the so-called Zombie list[35,36]. When the two packets belong to the same flow (for example, they have the same destination and source addresses, the same destination and source port numbers and the same protocol), a hit is declared and the count is incremented in the corresponding entry

in the Zombie list. Otherwise the entry in the Zombie list is replaced with a probability p. Actually in the version of SRED proposed by the discard probability is entirely based on the instantaneous length of the buffer queue and on the number of active flows, but the adding of average queue size estimation does not present any difficulties. This improvement has the advantage to stabilize the buffer occupancy, independently of the number of active connections and to provide a way through the hit-mechanism to detect situations of unfairness, in which some flows are attempting to take more than their fair share of bandwidth (misbehaving flows).

3.3.6 GRED

GRED is an improvement of RED (Random Early Detection) proposed in. RED drastically changes the packet drop probability to one when the average queue length is large. Hence , when the average queue length is large, the queue length become unstable. GRED prevents the queue length from becoming unstable by gently changing the packet drop probability .In what follows,we briefly explain the algorithm GRED. The packet dropping algorithm of GRED is essentially the same as that of RED.

GRED[37] maintains the average queue length as well as RED. For every packet arrival, the average queue length q is updated as:

$$\overline{q} \leftarrow (1 - w_q)\overline{q} + w_q q$$

where q is a current queue length, and wq is one of GRED's control parameters, which specify the weight of an exponential averaging. GRED determines the packet drop prob-

ability *pb* based on the average queue length *q* as:

$$p_b = \begin{cases} 0 & \text{if } \overline{q} < min_{th} \\ max_p(\frac{\overline{q}-min_{th}}{max_{th}-min_{th}}) & \text{if } min_{th} \leq \overline{q} < max_{th} \\ (1-max_p)(\frac{\overline{q}-max_{th}}{max_{th}}) + max_p & \text{if } max_{th} \leq \overline{q} < 2\,max_{th} \\ 1 & \text{if } \overline{q} > 2\,max_{th} \end{cases}$$

where *minth* is the minimum threshold, *maxth* is the maximum threshold, *maxp* is the maximum packet drop probability, and all are control parameters of GRED. GRED randomly drops an arriving packet with the probability *pa* defined by

$$p_a = \frac{p_b}{1 - count \times p_b}$$

where *count* is the number of packets that have arrived at the router since the last packet dropping

3.3.7 DRED (Dynamic RED)

RED has a problem that the average queue length is dependent on the number of active TCP connections. DRED solves this problem by using the feedback control, which adjusts the packet drop probability in proportion to its average queue length. DRED is therefore able to stabilize the queue length at the target value without being dependent on the number of TCP connections.

We briefly explain the algorithm of DRED. DRED uses a fixed sampling interval, and the packet drop probability is updated every sampling interval. In what follows, we focus on the packet that arrives at the router in the *n*th sampling interval. First, DRED obtains the error signal as

$$e(n) = q(n) - T$$

Next, the filtered error signal of $e(n)$(denoted by $e\hat{}(n)$) is updated as

$$\hat{e}(n) = (1-\beta)\hat{e}(n-1) + \beta e(n)$$

where β is the DRED's control parameter, and specifies the weight of an exponential averaging. Finally, using $e\hat{}(n)$, DRED determines the packet drop probability $pd(n)$as

$$p_d(n) = min\left[\max\left\{p_d(n-1) + \alpha\frac{\hat{e}(n)}{B}, 0\right\}, \theta\right]$$

where B is the buffer size of the router, α is the DRED's control parameter specifying the feedback gain of the packet drop probability, and θ is the maximum of the packet drop probability. The packet drop probability pd is updated every sampling interval, but DRED does not drop a packet if $q(n) < L$ for maintaining high resource utilization.

3.3.8 SRED (Simple SRED)

In RED, the average queue length depends on the number of TCP connections. Moreover, RED does not distinguish misbehaving TCP flows, which will not reduce their transmission rates after packet losses. For solving these problems, SRED estimates the number of active TCP connections in a statistical manner, and determines the packet drop probability according to the estimated number of TCP connections. For preventing unfairness caused by misbehaving TCP flows, SRED uses a different (i.e., large) packet drop probability for misbehaving TCP flows.

For estimating the number of active TCP connections, SRED uses "zombie list". The

zombie list maintains information on each TCP connection, and its size is denoted by *list*. Namely, each entry of the zombie list consists of a flow identifier, a counter, and a time stamp. When a packet arrives at the router, SRED compares a randomly chosen entry from the zombie list with the entry corresponding to the arriving packet. If these entries coincide, the counter in the entry is increased by one. Otherwise, the entry is probabilistically replaced by the information on the arriving packet with probability p. With the zombie list, SRED estimates the number of active TCP connections. For distinguishing misbehaving TCP flows, the zombie list is also used. See for the details of SRED.

We briefly explain the packet dropping algorithm of SRED. First, SRED compares a randomly chosen entry from the zombie list with the entry corresponding to the arriving packet. We focus on the nth arriving packet. If these entries coincide, $H(n)$is set to one. Otherwise, $H(n)$is set to zero. The probability $P(n)$ that the zombie list contains the entry for the arriving packet is estimated by

$$P(n) = (1-\alpha)P(n-1) + \alpha H(n)$$

where α is the SRED's control parameter, and specifies the weight of an exponential averaging. Next, in proportion to the current queue length q, the packet drop probability psred(q)is updated for every packet arrival as

$$p_{sred}(q) = \begin{cases} p_{max} & \text{if } \frac{1}{2}B \le q < B \\ \frac{1}{4} \times p_{max} & \text{if } \frac{1}{6}B \le q < \frac{1}{3}B \\ 0 & \text{if } 0 \le q < \frac{1}{6}B \end{cases}$$

where B is the buffer size of a router. pmax is the SRED's can be estimated from the

regression coefficients. control parameter, and limits the maximum of the packet drop probability. Finally, SRED randomly drops an arriving packet with the probability pzap defined by

$$p_{zap} = p_{sred}(q) \times min\left(1, \frac{1}{(256 \times P(n))^2}\right) \times \left(1 + \frac{H(n)}{P(n)}\right)$$

3.3.9 Flow RED

FREDallows each connection to buffer *minq* packets and apply discard probability to the subsequent packets. It never permits a flow to buffer more than *maxq* packets and it stores the times in which it has tried to do that in *strike*. Flows with high *strike* are more subject to loss packets and they cannot buffer more than *avgcq* packets. Moreover, in FRED the frequency of *avg* calculation is higher than in RED. In fact, the averaging is done at both arrival and departure in order to provide a more accurate estimation of *avg,* and when a packet is dropped *avg* does not vary. Simulation results show that FRED[38] is often fairer than RED when handling connections with different round trip times and window sizes and it allows detecting unresponsive users. Anyway, it adds overhead to store information about the flows which have packets buffered in the router. The cost of this per-active-flow accounting is proportional to the buffer dimension and independent of the total number of connections served by the gateway.

3.3.10 Adaptive RED

Adaptive RED focuses on the problem of parameterizing the RED algorithm in order to reach good performance in each possible scenario[39]. For example, in the case of a bottleneck link shared by N connections, congestion notification requires each connection to reduce the traffic of $\left(1-\frac{1}{2N}\right)$. If N is large the effect of traffic reduction by each connection will be small, and on the contrary, if N is small it will be considerable. In the first case we need a more aggressive RED algorithm in order to avoid packet loss and to perform as a simple Tail Drop queue; in the second case we need a less aggressive algorithm to keep the link utilization to an acceptable level. Hence there are two main drawbacks in using RED[40]:

- The average queueing delay with RED is sensitive to the traffic load and to parameters, and consequently is not predictable in advance. When the congestion is light and/or *maxp* is high, *avg* is close to *minth* and when the congestion is heavy and/or *maxp* is low, the *avg* is near to *maxth.*. In order to have some guaranteed delay with RED it is necessary to perform a frequent tuning of its parameter according to the traffic variations.
- The second point is that the throughput is also sensitive to the traffic load and to the parameters. In particular when the *avg* is larger than *maxth* the throughput performance decreases greatly.

The solution proposed is to provide an adaptive variation of RED parameters based on the *avg*. The key idea is to adapt *maxp* (the initial discarding probability parameter) in

order to keep the average queue size between *minth* and *maxth*. When *minth£ avg £ maxth* there are no variations, but if *avg* < *minth*, RED must be less aggressive and *maxp* = *maxp*+a. Otherwise (*avg*>maxth) RED must be more aggressive and *maxp* is increased: *maxp* = *maxp*+ b. a and b are constant factors. A pseudo-code describe :

```
Every interval seconds:
   if (avg > target and max_p ≤ 0.5)
      increase max_p:
      max_p ← max_p + α;
   elseif (avg < target and max_p ≥ 0.01)
      decrease max_p:
      max_p ← max_p * β;

Variables:
avg:  average queue size

Fixed parameters:
interval:  time; 0.5 seconds
target:  target for avg;
   [min_th + 0.4 * (max_th − min_th),
      min_th + 0.6 * (max_th − min_th)].
α:  increment; min(0.01, max_p/4)
β:  decrease factor; 0.9
```

Figure 3.6 adaptive RED pseudocode

figure 3-6 adaptive RED pseudocode

3.3.11 RED with Penalty Box

It is one of the first proposals intended to distinguish responsive users from unresponsive users. It is based on the observation that high bandwidth flows see proportionally larger amounts of packet loss. It maintains a list of the recent packet loss events verified in the network in order to identify all the misbehaving flows in the penalty box. These unresponsive flows are limited in the rate using a mechanism such as

class-based queueing.

3.3.12 GREEN

The GREEN algorithm is a feedback control function which adjusts the rate of congestion notification in response to the flow based congestion measure, *xest*, the estimated data arrival rate. GREEN is based on a threshold function. If the link's estimated data arrival rate *xest* is above the target link capacity cl, the rate of congestion notification, *P*, is incremented by Δ P at a rate of 1/Δ T. Conversely, if *xest* is below cl, *P* is decremented by Δ P at a rate of 1/Δ T[41,42,43]. The algorithm applies probabilistic marking of incoming packets at the rate *P*, either by dropping packets, or setting the ECN. Let the step function U(x) be defined by:

$$U(x) = \begin{cases} +1 & x \geq 0 \\ -1 & x < 0. \end{cases}$$

Therefore:

$$P = P + \Delta P \cdot U(x_{est} - c_t).$$

the actual link capacity *c*, typically 0.97 *c*, so that the queue size converges to 0. Incoming data rate estimation is performed using exponential averaging:

$\hat{x}_{est}$ = (1-exp(-*Del*/*K*))*(*B*/*Del*)+exp(-*Del*/*K*)* *xest* (5)

where Del is the inter-packet delay, *B* the packet size and *K* the time constant. Other arrival rate estimation techniques could also be used successfully. In Eq $\hat{x}_{est}$ is shown to converge to the actual data rate under a wide set of conditions.

3.3.13 CHOKe（CHOose and Keep for responsive flows CHOose and Keep for unresponsive flows）

CHOke is another proposal, based on the RED algorithm, whose goal is to approximate fair queueing and be[44], at the same time, simple to implement. It is based on the assumptio n that the FIFO buffer is a reliable indication of which flows are consuming a great amount of resources. If a packet arrives at a congested router (*avg* > *minth*), if the *avg* > maxth, the packet is discarded, like in normal RED, otherwise the new packet is compared to a randomly chosen packet from the FIFO buffer. If they belong to the same flow they are both discarded, otherwise the randomly drawn packet is left intact and the new one is admitted into the FIFO buffer with a probability calculated like in the original RED, based on the avg[45,46]. The advantage of this algorithm compared to those mentioned above is that it does not need any state information and consequently it introduces the minimal implementation overhead.

3.3.14 RIO（RED with In and Out）

It is based on the two-drop precedence policy. A packet is marked at the edge of the network as IN or OUT of its service contract and it is treated differently inside the network, on the basis of this priority classification. The router inside the network keeps just one queue for IN and OUT packets and apply to them two different RED algorithms. Instead of using the same average queue size for both priorities, it uses the average queue size for OUT (out of profile) packets, and the average queue size without taking into

account the queued OUT packets for IN (in profile) packets[47,48,49]. In time of congestion the router starts to drop OUT packets and eventually, if congestion persists, will start to discard IN packets, as well.

3.3.15 Weighted RED

WRED was initially proposed as RIO. It has been developed as an extension of the RED approach by taking into account the priority of packets[50]. It assigns to every priority a different RED algorithm, making it possible to differentiate the performance of different TCP connections whose packets are queued in the same queue. Different QoS can be provided for different classes. For example packets from higher priority traffic is dropped with a lower probability than the standard traffic, during periods of congestion. This is implemented with two RED algorithms running in parallel. In order to reduce the packet loss rates experienced with RED a different queue management algorithm has been implemented.

Finally, we use a table to find out the advantage and disadvantage of these AQM algorithms.

AQM algorithm	advantage	disadvantage
RED	congestion avoid congestion control Avoid the global synchronization	Fairness problem Have the priority problem parameter sensitivity when flux load happen to change ,the

	phenomenon	fixed value MAXp will arouse queue shock
ECN	detects congestion and sends echo message to the end nodes	Send to many echo to end nodes possibly
BLUE	uses packet loss (queue) and link under-utilization (flow) events to adjust the rate of congestion notification early little buffer	greater the RTT long term queue length average produces slow response can not avoid some degree of multiple packet loss and/or low utilization
SFB	Using charge account to limit the non-adaptive flow`s speed	difficulty configure parameters
SRED	stabilize the buffer occupancy, independently of the number of active connections and to provide a way through	Some Per-flow state Red disadvantage

	the hit-mechanism to detect situations of unfairness	
Stabilized RED	Keep fifo queue stabilize Distinguish non-adaptive flow	$P(i)^{-1}$ is not a good estimator for heterogeneous traffic Parameter tuning problem Stabilize queue occupancy when traffic load is high
FRED	per-active-flow keep accounting slove fairness question	Per-flow state Red disadvantage
Adaptive RED	Adapt max_p based on queue behavior	Red disadvantage
GREEN	Adjust to send congestion message againest congestion degree	Send to many echo to end nodes possibly
CHOKe	Protcet adaptive flow and punish non- adaptive flow	Some complexity to due to parameters Low throughput in some case
RIO	Interior reserved bandwidth is IN,over is	Parameter sensitivity

	OUT OUT have higher dropped rate	
Weighted RED	Have 8 priority of packets	Lower parameter sensitivity

TABLE 3-1 AQM ALGORITHM COMPARE

CHAPTER 4

AAQM DESIGN

4.1 Base theory

First ,we will introduce to negative feedback control in biology. In animals such as ourselves, the internal environment of our bodies must have certain conditions within tolerable limits to continue the healthy functioning of us.

This is done by a process called negative feedback control, where various receptors and effectors bring about a reaction to ensure that such conditions remain favourable. For this control, we investigate the control of blood sugar concentrations, water concentrations and temperature.

The principle of negative feedback control is illustrated by the diagram below

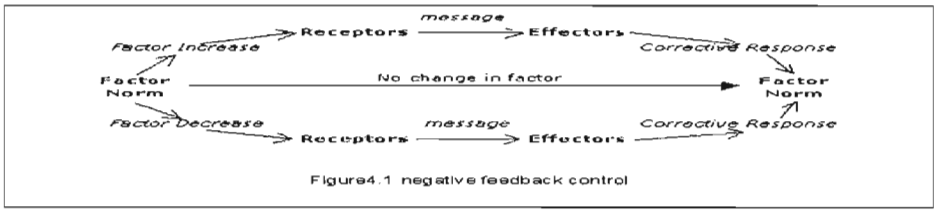


figure 4-1 negative feedback control

This occurrence is known as physiological homeostasis, translating in layman's terms to the physical equilibrium. It is essentially a corrective mechanism, consider the following scenario in a person

4.2 Example

The adrenal glands synthesize three main classes of hormones : mineralocorticoids, glucoconicoids, and sex steroids. Figure4.2 shows a simplified scheme of the adrenal synthesis of these steroids from the cholesterol precursor molecule. Each enzyntatic step is indicated.

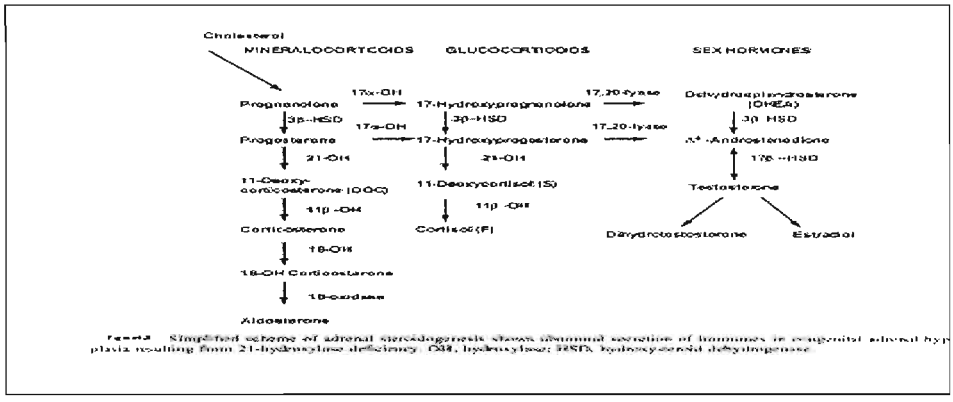


figure 4-2 simplified scheme of adrenal steroidogenesis shows abnomal seretion of hormones in congenital adrenal hyperplasis resulting from 21-hyroxylase deficiency

The pituitary regulates adrenal steroidogenesis via adrenoconicotropic hormone (ACTH). ACTH stimulates steroid synthesis by acting on the adrenals to increase the conversion of cholesterol to pregnenolone, which is the principal substrate for the

steroidogenic pathways. The central nervous system controls the secretion of ACTH, its diurnal variation, and its increase in stress via corticotropin-releasing factor. The hypothalamic-pituitary-adrenal feedback system is mediated through the circulating level of plasma cortisol; any condition that decreases cortisol secretion results in increased ACTH secretion. Cortisol therefore have a negative feedback effect on ACTH secretion.

In most forms of congenital adrenal hyperplasia, an enzyme defect blocks conisol synthesis, thus impairing cortisol-mediated negative feedback control of ACTH secretion (Fig . 2) . Oversecretion of ACTH ensues, which stimulates excessive synthesis of the adrenal products of those pathways unimpaired by an enzyme deficiency and causes an accumulation of precursor molecules in path- ways blocked by an enzyme deficiency.

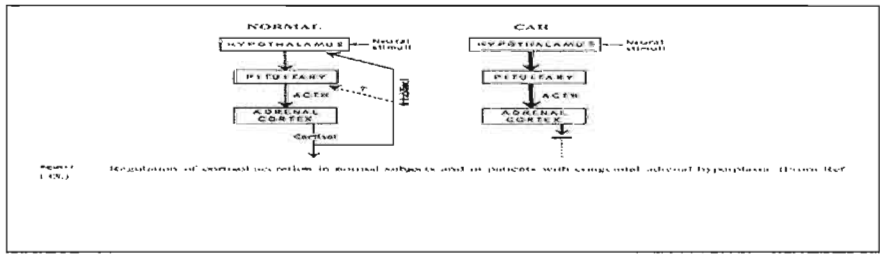


figure 4-3 Regulation of cortisol secretion in normal subjects and in pateints with congenital adrenal hyperplasia

From upper figure 4.3 ,we can see how to control cortsal in normal body . When cortisal concentration reduce to min_{th} threshold value, hypothalamus will accept neural stimulate to increase corticotropin releasing hormone (CRH) concentration. CRH will arise ACTH concentration. At last ACTH will arise cortisal concentration. On the other hand, cortisal concentration exceed to max_{th} threshold value, hypothalamus will accept

neural stimulate to decrease corticotropin releasing hormone (CRH) concentration. CRH will reduce ACTH concentration. At last ACTH will reduce cortisal concentration. Under CRH-ACTH-AD axis controled, cortisal can keep a shake variational curve in normal area like RED simulate result. I use this idea to design a ECN message-token bucket-packet transporttion axis. There are two available cause: (1) packet transport also have min_{th} and max_{th} threshold value. (2) tokens have update mechanism like ACTH diurnal variation. When avg exceed min_{th} threshold value, AAQM begin to pick packet for early congestion notification (ECN). The packets is then marked, so ECN message count will increase. Tokens will incresase more by update itself. At last packet transporttion will increase. When avg exceed max_{th} threshold value, more packets will force to drop and ECN message count will reduce. At last packet transporttion will reduce because of network congestion.

4.3 AAQM

The new adaptive queue management algorithm is a improved RED algorithm based on adaptive negative feedback control. As more research is conducted on adaptive active queue management, the different mechanisms in adaptive queue management trends to be divided separately. Though some similar view has been expressed in, it is necessary to describe it more clearly here. All mechanisms used in adaptive queue management must research three components[51]:

(1)Congestion measure function is used to measure the degree of congestion. It tries to

return a value with available information to indicate how sever or lighten the current congestion status. From the evolution of congestion measure function, we can see it becomes more and more complicated. With the introduction of more information and more processing to the information, the congestion measure functions provide congestion indication more and more precisely.

(2)Congestion feedback function is used to calculate how much feedback should be sent to the end system. It uses the output of congestion measure function as input and returns the probability for taking action.

(3) Congestion feedback action Congestion feedback action determines what action will be taken with the probability calculated by the congestion feedback function. So we describe the technique way by this way.

AAQM is an improved RED algorithm. In RED algorithm, set the average queue length is avg to limit the max_{th} up and down with the min_{th}. When avg is smaller than min_{th} , do not throw away the arrived packet. Now we define a new threshold queue length q(assumption set up for 75% max_{th}) here, when queue value is between the avg and max_{th}. Triggering two course : (1) the router sends out (call the IN port) a segment deceleration requests to source end, the decelerating claim have the different deceleration to the different queues. When the threshold queue length increase 5% min_{th} the router send a deceleration requesting to source end . (2) In the figure4.3, the router send flow to destination end (call the OUT port) and use AAQM algorithm to proceed traffic shaping. At beginning, r denotes the rate at which tokens are accumulated ,ptsize is packet

length,and b is the depth of the token pool(in bytes). When the average queue length is smaller than threshold q in any interval t [S,S+t], the tokens is b+(r-ptsize)t ; when average queue length avg is bigger than q in any interval t [S,S+t], the tokens is b+(2r-ptsize)t to promise enough packets can be transmitted . When the average queue length avg is bigger than $max_{th,}$ this algorithm begin to drop packet as RED .The course (1) and (2) are both feedback course , but the feedback length of course(2) is shorter than course(1) . When average queue length avg is between the threshold and max_{th}, course(1) get up the effect quickly than course(2) .Two kinds of feedback common operation can guarantee avg is smaller max_{th} in length and rise to precaution network congestion. This method can try to fall the value of average queue length avg quickly. When the average queue length avg trecovers smaller than threshold queue length, the tokens recovers b+(r-ptsize)t . This kind of many cache classes mechanism can postpone the occurrence of network congestion consumedly and guaranteed the degree of resources usage. This is a kind of virtual circuit feedback algorithm.

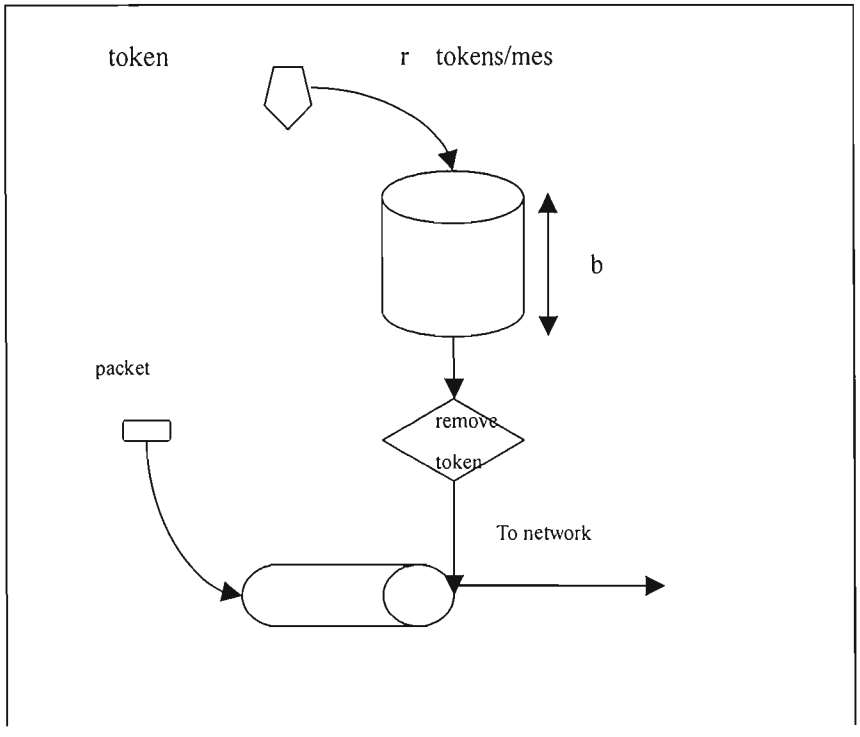


figure 4-4 AAQM mechanism design

Now we can realize the new algorithm in figure4.4 .After enque and deque packets ,queue will be transmitted. This process is controlled through tokens quantity. If bucket have enough tokens to be used , the packets in the queue will be sent at once.

```
enque();//whether mark CE=1 or not

deque();

//This is to update the tokens which used for the next action.

tokens_ = getupdatedtokens();

//Now start token and ptsize and prepare the transmistion

if (tokens_ >=pktsize)

target_->send();

tokens_-=pktsize;

else

target_->send(tokens);

tokens_=0;
```

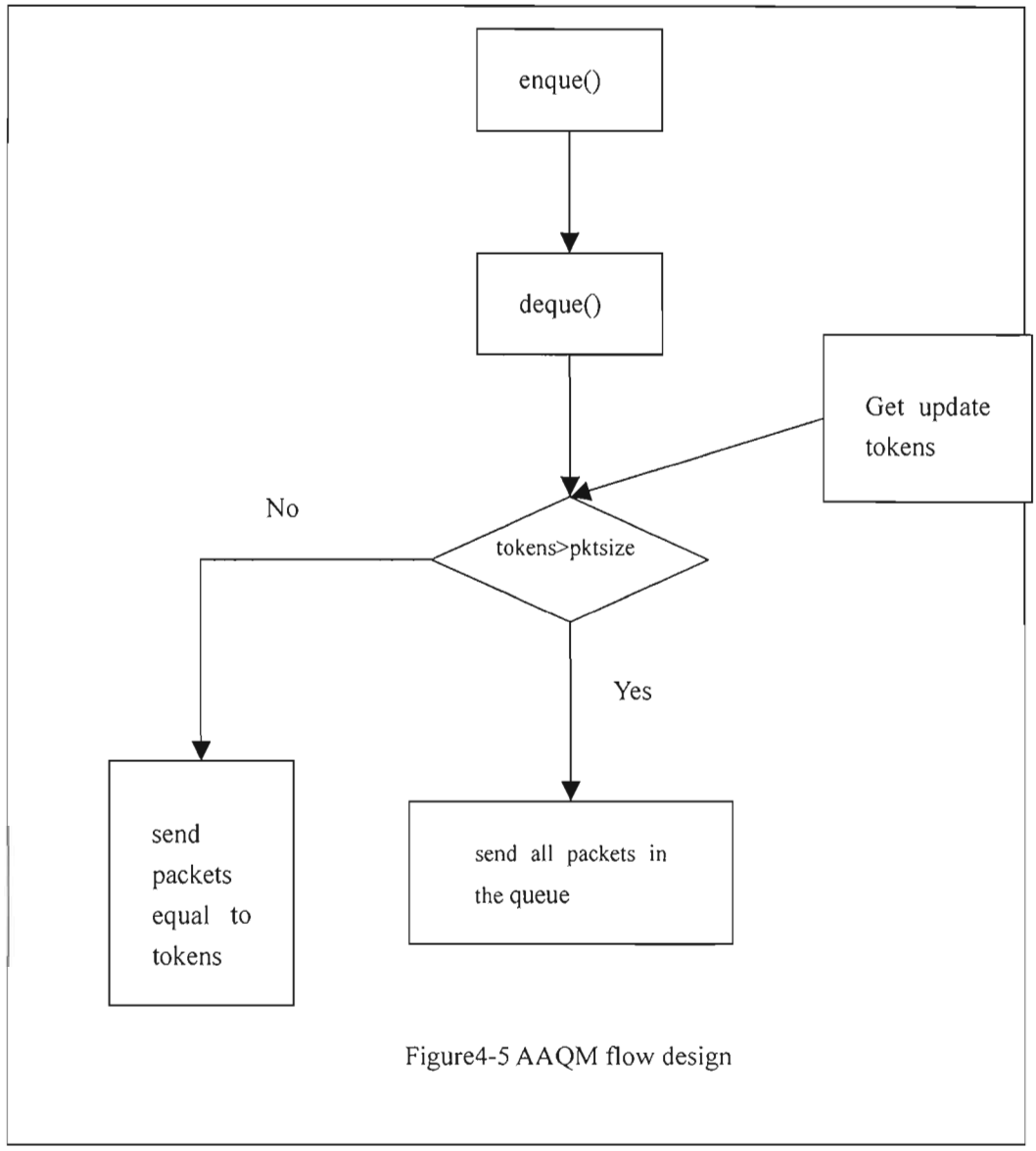


Figure4-5 AAQM flow design

figure 4-5 AAQM flow design

CHAPTER 5

PERFORMANCE EVALUATION AND SIMULATION RESULTS

A number of simulations were performed to compare the performance of AAQM and RED using the NS-2 simulator. The ns Network Simulator (current version ns-2) is an event driven simulator for computer networks and network protocols. Since it is widely used in and contributed to by the research community, a large number of network components are available for ns. Amongst others, ns supports the following technologies:

- Point-to-point connections
- Different router discard strategies (DropTail, RED, etc.)
- TCP, UDP, and several experimental transport protocols
- Applications (Telnet, FTP, WWW-like traffic, etc.)
- Network emulation (i.e. interaction of the network simulator with a "real" operating network node)

The simulator framework uses a split-language programming approach. The simulator core, including the actual protocol implementations, is written in C++, while the control structure and the description of simulation scenarios is done in OTcl, an object oriented version of Tcl. This approach utilizes the flexibility advantages of a scripting

language for scenario creation while taking advantage of the better efficiency of compiled code where necessary.

Any useful ns scenario consists of the following components:

- The topology
- Communication Patterns
- Events

The topology is made up of nodes with queues and links between the nodes, and defines network conditions such as the link bandwidth, delay, random losses, and queue sizes. Communication Patterns determine what kinds of traffic are transmitted between the nodes. Common events are starting and stopping a flow, or changing the random loss rate on a link.

A protocol implementation for ns mainly consists of two objects representing sender and receiver, with methods deciding when to send what kinds of packets and how to react to incoming packets.

The network topology and parameters common to all trials are described here.

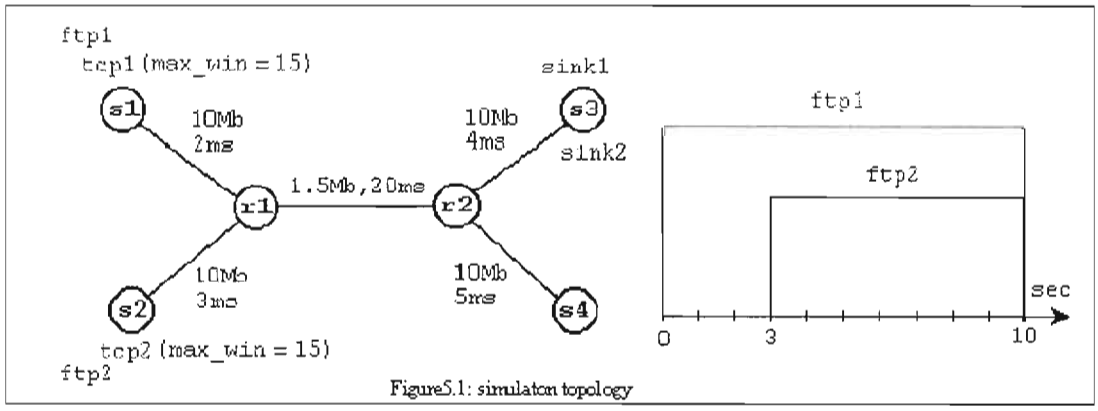


Figure5.1: simulaton topology

figure 5-1 simulaton topology

The left nodes are the sources and the right nodes are destination. The AQM and RED algorithm is placed in center link(r1,r2). Each trial lasted 10 seconds. Connections were started at a uniformly distributed time over the test period with a uniformly distributed duration between 0 and 10 seconds. The packet size was 500 bytes. The default configuration parameters for each of the AQMs followed recommended settings:

RED: minThreash=5 , maxThreash=15, W=0.002

5.1 Trial 1

Setting:

bottleneck : 1.5mb , 20ms ,100 packet queue size which have Red or AAQM algorithm

Red: minThreash-5, maxThreash-15, w=0.002

AAQM: rate_= 64k bucket_= 30000 qlen_= 60

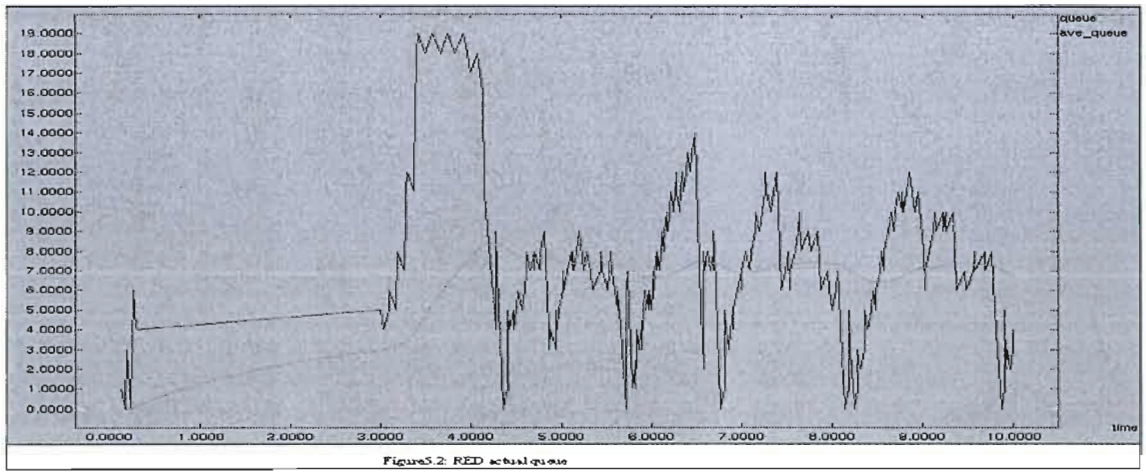


Figure5.2: RED actual queue

figure 5-2 RED actual queue

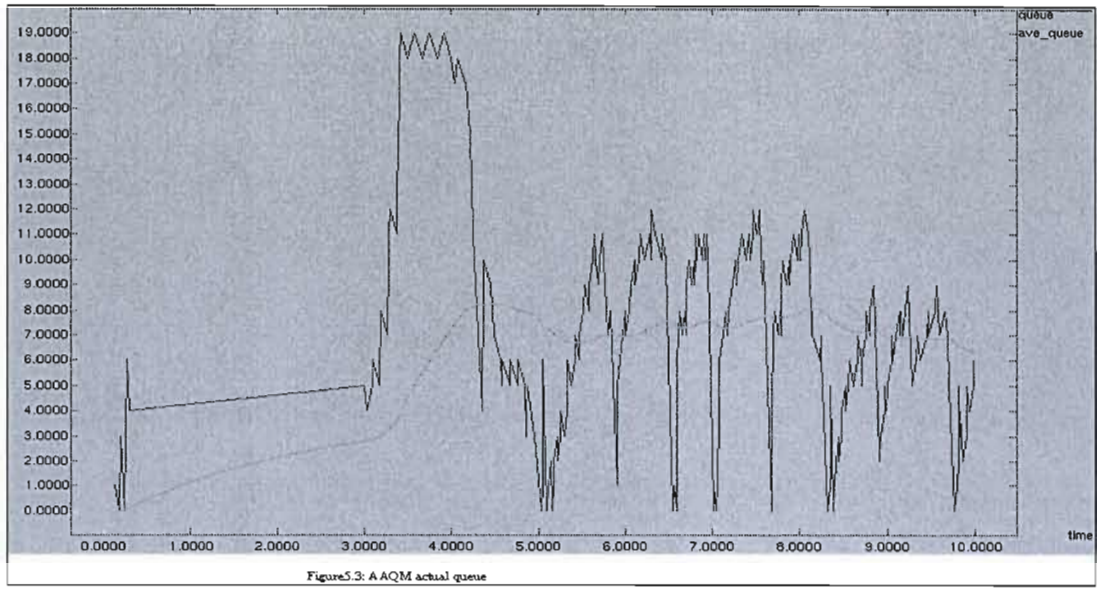


figure 5-3 AAQM actual queue

The upper figure shows RED algorithm results and the lower figure shows AAQM algorithm. AAQM algorithm results shows lower queue variation than RED algorithm results. This means AAQM algorithm can get better link utilization and make lower queuing delay. But the estimated average queue size of RED and AAQM grows slower than actual queue size. This slow phase effect of estimated average queue length makes AQM algorithms response slowly to the congestion. This phenomenon appears in every queue size based AQM algorithms.

5.2 Trial 2

Default Setting:

bottleneck: 1.0mb , 50ms ,100 packet queue size which have Red or AAQM

algorithm

Red: minThreash-5, maxThreash-15, w=0.002

AAQM: rate_= 64k bucket_ =30000 qlen_ =60

For each simulation we changed bandwidth size from default setting. we changed bandwidth from 1.0 M to 3.0M increasing 0.5 per each.

From trail 1, AAQM or RED begin to deal with congestion when queue length quickly increase to max queue length. We can see network transportations in normal status before this time. Under figure5.4, AAQM has longer time than RED in the below figure(first max queue time VS bandwidth). In AAQM ,We estimate AAQM have enough tokens in bucket to use transportation at beginning. With transportation rate increase and tokens update ,Tokens and transported packets are in an active balance status. When this balance is broken, networks begin to congestion. Making use of well precongestion phases can increase the network transportation.

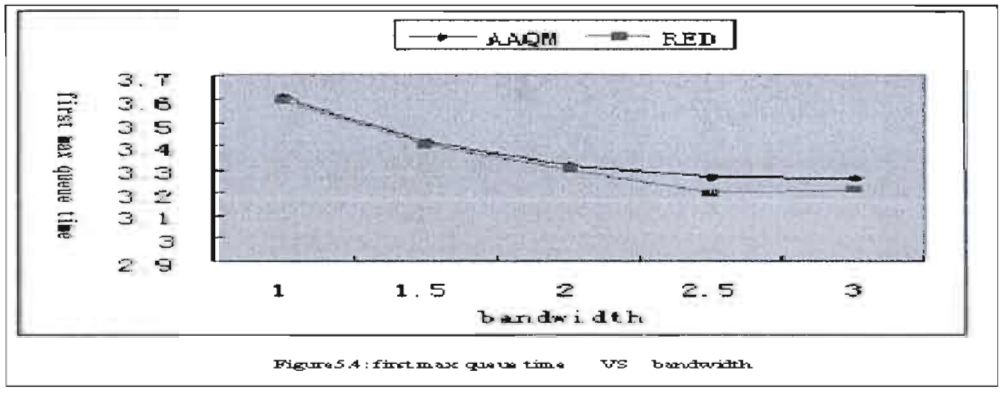


Figure5.4 : first max queue time VS bandwidth

figure 5-4 first max queue time VS bandwidth

5.3 Trial 3

Default Setting:

bottleneck: 1.5mb , 20ms ,60 packet queue size which have Red or AAQM algorithm

Red: minThreash-20% of queueLimit, maxThreash-80%of queueLimit, w=0.002

AAQM: default

For each simulation we changed Max queue size from default setting. we changed Max queue size from 20 packets to 140 packets increasing 20 per each. AAQM has lower packet loss rate than RED in the below figure(max queue limit VS packet loss rate).

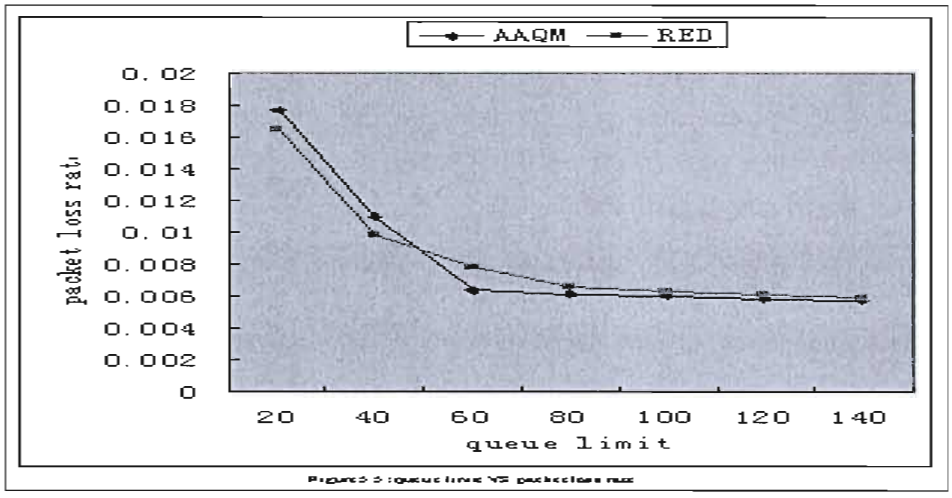


figure 5-5 queue limit VS packet loss rate

Next figure shows Goodput VS max buffer size(queue limit). Goodput is the ratio of the total number of nonduplicate packets received at all destinations per unit time to link capacity. If loss rate is p, total number of nonduplicate packet is node utilization * link

capacity * (1-p −p^2 −p^3 ⋯) = node utilization * link capacity * (1 −p). In next figure, Goodput was calculated from Gooput = node link utilization * (1-p). From below figure we can know AAQM has better Goodput than RED.

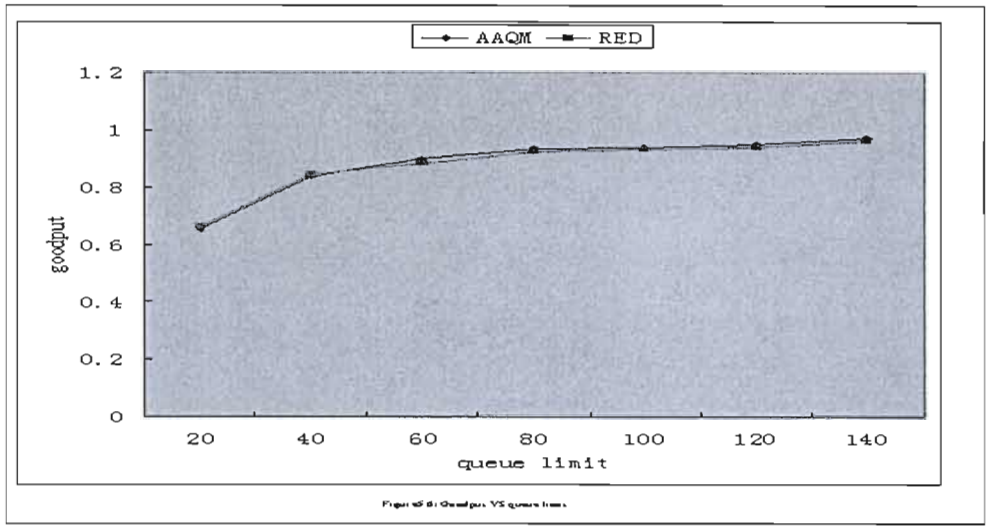


figure 5-6 Goodput VS queue limit

Below figure shows max buffer size(queue limit size) VS average queuing delay. AAQM have larger delay than RED. Because AAQM has lower loss rate and larger link utilization, it has slightly larger average queue size.

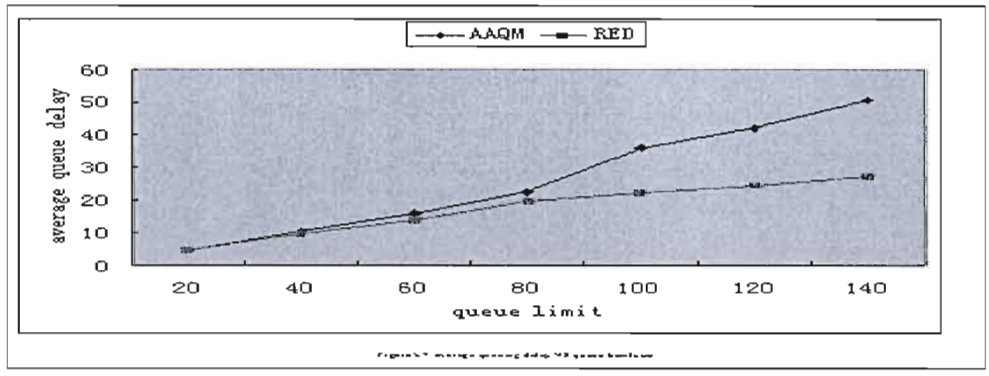


figure 5-7 average queuing delay VS queue limit size

5.4 Trial 4

Default Setting:

bottleneck router: 10mb , 50ms ,100 packet queue size which have Red or AAQM algorithm

Red: minThreash-20% of queueLimit, maxThreash-80%of queueLimit, w=0.002

AAQM: default

FLOW : 25 FTPs with propagation delay uniformly distributed from 10ms to 160 ms.

We have changed TCP flow numbers from default setting. we changed FTP flow number from 25 to 150 increasing 25 per each.

Below figure shows number of TCP session vs average packet loss rate. AAQM has low loss rate than RED.

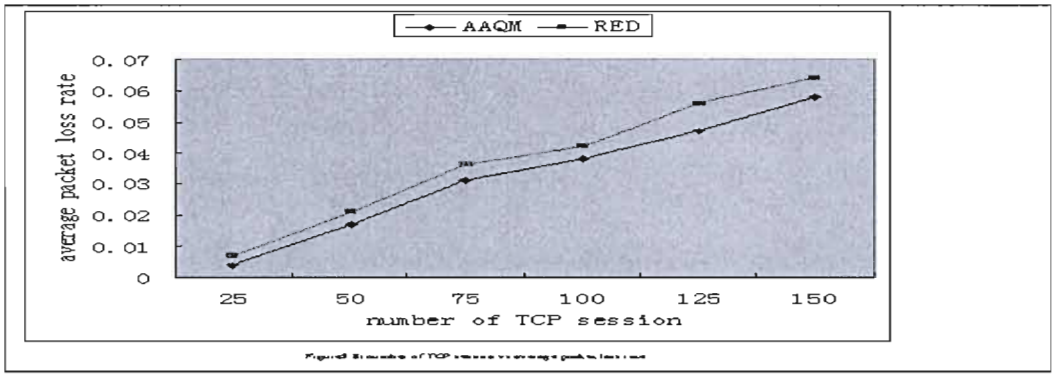


figure 5-8 number of TCP session vs average packet loss rate

5.5 Trial 5

Default Setting:

bottleneck router: 10mb , 50ms ,100 packet queue size which have Red or AAQM algorithm

Red: minThreash-20% of queueLimit, maxThreash-80%of queueLimit, w=0.002

AAQM: default

FLOW : 100 FTPs with propagation delay uniformly distributed from 10ms to 160 ms.

For each simulation we changed Max queue size from default setting. we changed Max queue size from 20 packets to 140 packets increasing 20 per each.

AAQM has lower packet loss rate than RED in the smaller buffer size than 80 packet size.

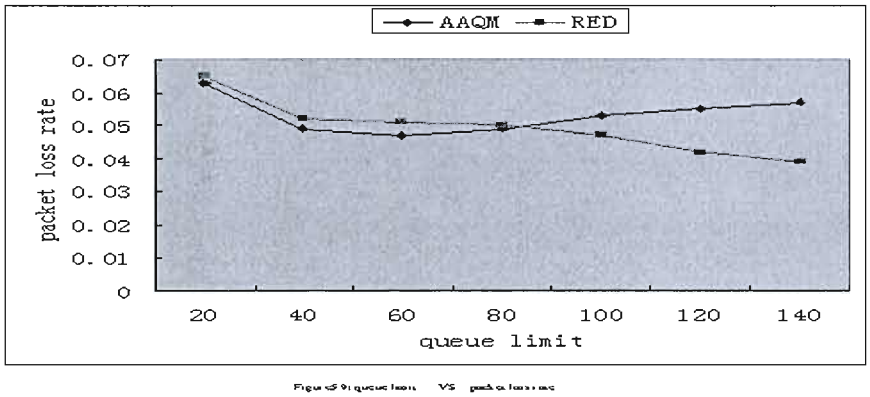


figure 5-9 queue limit VS packet loss rate.

Next figure shows Goodput VS max buffer size(queue limit). From below figure we can know AAQM has better Goodput than RED when buffer size is smaller than 80 . This is owing to that AAQM maintain target queue size in spite of larger buffer size. So AAQM algorithm does not decrease loss probability when it maintain target queue size

even if it has larger buffer size.

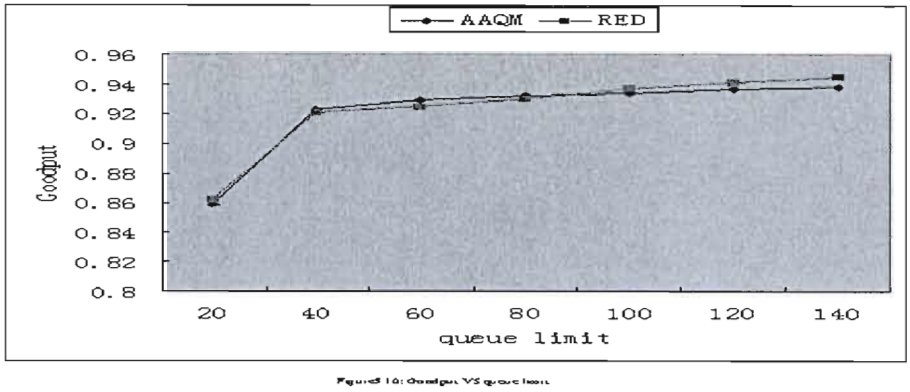


figure 5-10 Goodput VS queue limit

Below figure shows max buffer size(queue limit size) VS average queuing delay. AAQM algorithm's delay do not increase at certain degree. . This is owing to that AAQM maintain target queue size in spite of larger buffer size. So AAQM algorithm does not decrease loss probability when it maintain target queue size even if it has larger buffer size.

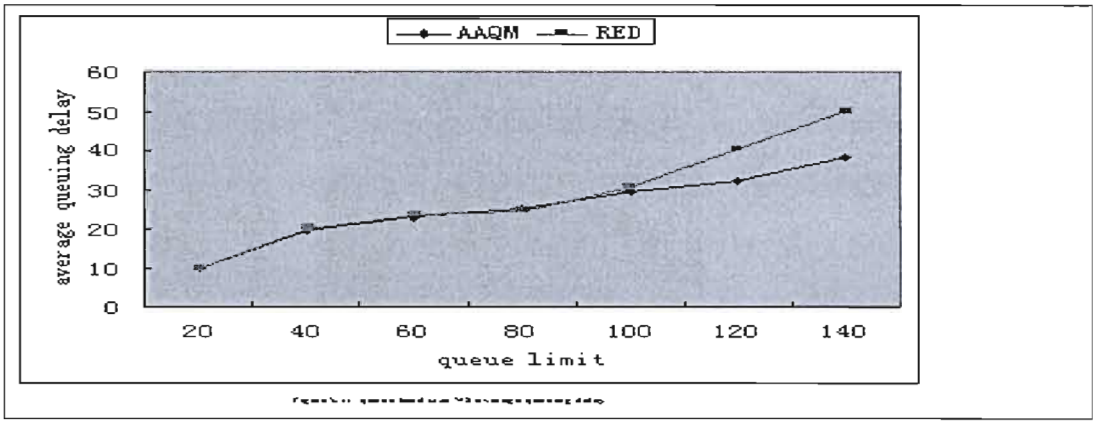


figure 5-11 queue limit size VS average queuing delay

CHAPTER 6

CONCLUSION AND FUTURE WORK

In the present thesis I have analysed the RED and AAQM active queue management algorithm and verified how much effective mechanism is for congestion avoidance at the ns-2 simulator. The expected benefits derived from its employment have been evaluated by observing the effects produced on TCP service performance. Essentially my analysis is based on a comparative study between AAQM and RED mechanisms.

The main problem the TCP traffic have to face up to in such an environment is the heavy congestion simulated on the bottleneck router where the bandwidth is strictly limited. We have presented the RED algorithm and derived some guidelines in tuning its parameters. I have made an overview of Active Queue Management scheme mentioning several related studies and suggested AAQM which improves to RED mechanisms. In particular I have introduced ECN as one of the most promising techniques to be employed in combination with RED.

The first relevant result of my tests concerns the expected good AAQM results of network. Our experiments demonstrate that AAQM was shown to maintain a lower packet loss rate and lower delay as well as better goodput. The algorithm is robust and

easily configured .

The second result concerns the AAQM and RED issue of minimizing the number of packets dropped in the node. With regards to this aspect, RED fails since all our RED tests experience high rates of retransmitted packets. However the results show that even if RED discards the large number of packets retransmitted does not degrade the performance, resulting on the contrary in improved goodput for the network. I have observed that by starting to discard packets before the buffer queue overfills, RED prevents severe congestion states. It manages to absorb transient congestion, such the one caused by competing flows or by drastic reduction in the available bandwidth due to the arrival of higher priority traffic. The RED recovery phase has proved to be fast even though multiple segments are dropped. Especially if packets are lost in some critical moments, such as the connection ending. Packets are discarded in bursts and the recovery phase takes a long time. RED starts to discard early and randomly. The retransmitted packets are quickly acknowledged and avail of the SACK option. AAQM keeps the RED characteristic in minimizing the number of packets dropped .Using available tokens try to send out more packets than RED. This include the possible packets dropped by red marked, so AAQM has lower the number of packets dropped.

The third relevant result of our tests concerns the expected AAQM property of precongestion. Our experiments demonstrate that AAQM have enough tokens in bucket to use transportation at beginning. During transportation rate increasing and tokens updating , Tokens and transported packets are in an active balance status. When this

balance is broken, network begins to congestion phases. Making use of well precongestion phases can increase the network transportation.

The theory and experiments have demonstrated the ability of the AAQM algorithm to improve connection goodput and reduce packet loss. It is not our intention to claim that AAQM is optimal or better than RED, but our experiments show that it works for wide range of scenarios and provides good technique so that it can be perfected and deployed with RED implementations in the next generation Internet routers.

Future research is also needed to determine the optimum average queue size for maximizing throughput and minimizing delay for different network and traffic conditions. In my study we did not consider as performance metric the router queue length. It could be useful to develop a study to evaluate how aggressive the AAQM algorithm is and how much it is able to absorb and accommodate packets burst. On the other hand, AAQM is compared with other AQM algorithms such as BLUE,REM, GREEN.

Finally I can conclude that a deployment of the AAQM could make RED more attractive. By following this path of evolution RED will be able to provide advance warning of incipient congestion more efficiently and with less waste of resources.

Bibliography

References:

[1] V. Jacobson, "Congestion Avoidance and Control," *Proc. ACM SIGCOMM '88*, 1988, pp. 314–29.

[2] D. Scott Alexander, Jonathan M. Smith –The Architecture of ALIEN in "Active Networks"- Proceedings First International Working Conference, IWAN'99 - Berlin June 1999, Springer ISBN 3-540-66238-3

[3] JAIN, R., RAMAKRISHNAN, K., AND CHIU, D.-M. Congestion avoidance in computer networks with a connectionless network layer. Tech. Rep. DEC-TR-506, Digital Equipment Corporation, Aug. 1987

[4]Nagle, J., "Congestion Control in IP/TCP", RFC 896, January 1984.

[5]Floyd, S., and Fall, K., "Promoting the Use of End-to-End Congestion Control in t he Internet", IEEE/ACM Transactions on Networking, August 1999.

[6] T. Eguchi, H. Ohsaki, and M. Murata, "Multivariate analysis for performance evaluation of active queue management mechanisms in the Internet," in *Proceedings of SPIE's International Symposium on the Convergence of Information Technologies and Communications (ITCom 2002)*, pp. 144–153, July 2002.

[7] C. Lefelhocz *et al.*, "Congestion Control for Best-Effort Service: Why We Need a New Paradigm," *IEEE Network*, vol. 10, no. 1, Jan./Feb. 1996, pp. 10–19.

[8]S. Athuraliya and S.H. Low. "Optimization Flow Control, II: Random Exponential Marking", Submitted for publication, http://www.ee.mu.oz.au/staff/slow/research/, May 2000.

[9] R Mahajan and et al. "Controlling high bandwidth aggregates in the network". Technical Report, ACIRI, Feb 2001.

[10]Postel J. ed., *Internet Protocol,* RFC 791, September 1981.

[11]Postel J. ed., *Transmission Control Protocol,* RFC 793, September 1981.

[12] C. Hollot, V. Misra, D. Towsley, and W. Gong. On designing improved controllers for aqm routers supporting tcp flows. In *Proceedings of IEEE INFOCOM*, Apr. 2001.

[13] S. Floyd, "A Report on Recent Developments in TCP Congestion Control," *IEEE Commun. Mag.*, vol. 39, no. 4, Apr. 2001, pp. 84–90.

[14]C. Hollot, V. Misra, D. Towsley, and W. Gong, "On designing improved controllers for AQM routers supporting TCP flows."

[15]*with a Connectionless Network Layer: Concepts, Goals and Methodology*, Proc. IEEE Comp. Networking Symp., Washington D. C., April 1988, pp.134-142.

[16] Allman M., Paxson V. and Stevens W., *TCP Congestion Control,* RFC 2581, April 1999.

[17] Mathis M., Mahdavi J., Floyd S. and Romanow A., *TCP Sele ctive Acknowledgments Options*, RFC 2018, October 1996.

[18] Low, S.H, "A Duality Model of TCP and Queue Management Algorithms, " Proceedings of ITC Specialist Seminar on IP Traffic Measurement, Modeling and

Management, September 18-20, 2000, Monterey, CA.

[19] J. Aweya, M. Ouellette, and D. Y. Montuno, "A control theoretic approach to active queue management," *Computer Networks*, vol. 36, pp. 203–235, 2001.

[20]Braden, B., et al., "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998. [RFC2309]

[21] S. Floyd, "Recommendations on using the gentle variant of RED," May 2000.

[22]S.Floyd, and Jacobson, V., Random Early Detection gateways for Congestion Avoidance, IEEE/ACM Transactions on Networking, V.1 N.4, August 1993, pp. 397-413.

[23] P. Ranjan, E. H. Abed, and R. J. La, "Nonlinear instabilities in TCP-RED," in Proc. IEEE INFOCOM, 2002.

[24] Y. Zhang and L. Qiu, "Understanding the End-to-End Performance Impact of RED in a Heterogeneous Environment," Technical Report 2000-1802, Cornell University, Jan. 2000.

[25] Floyd S. and Jacobson V., *Random Early Detection gateways for Congestion Avoidance*, IEEE/ACM Transactions on Networking, V.1 N.4, August 1993, pp. 397-413.

[26] Floyd S., *Discussion on setting RED parameters,* November 1997.

[27] K. Ramakrishnan "The Addition of Explicit Congestion Notification (ECN) to IP" RFC3168

[28] P. Key, K. Lavens, and D. McAuley. An ECN-based end-to-end congestion-control

framework: experiments and evaluation. Technical Report MSR-TR-2000-104, Microsoft Research, October 2000.

[29] T. Kelly. An ECN Probe-Based Connection Acceptance Control. *Computer Communication Review*, 31(3), July 2001.

[30] K. Ramakrishnan and S. Floyd, "A Proposal to Add Explicit Congestion Notification (ECN) to IP," IETF RFC2481, Jan. 1999.

[31] S. Floyd, "TCP and Explicit Congestion Notification," *ACM Comp. Commun. Review*, vol. 24, no. 5, Oct. 1994, pp. 10–23.

[32] W. Feng, D. Kandlur, D. Saha, K. Shin, "Blue: A New Class of Active Queue Management Algorithms" *U. Michigan CSE-TR-387-99*, April 1999.

[33] W. C. Feng, K. G. Shin, D. D. Kandlur and D. Saha, "The Blue Active Queue Management Algorithms,"IEEE/ACM Transactions on Networking, August 2002, Vol. 10, No. 4, pp. 513-528.

[34] W. Feng, D. Kandlur, D. Saha, K. Shin, "Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness", *in Proc. of INFOCOM 2001*, April 2001.

[35] T. J. Ott, T. V. Lakshman, and L. H. Wong, SRED: Stabilized RED, *Proceedings IEEE Infocom* 1999

[36]Ziegler, T., Fdida, S., Brandauer, C., Hechenleitner, B.: Stability of RED with Two-Way TCP Tra□c. In: Proceedings of IEEE ICCCN 2000 (3). (2000)

[37]Tomoya Eguchi," On Control Parameters Tuning for Active Queue Management

Mechanisms using Multivariate Analysis",

[38] T. Eguchi, H. Ohsaki, and M. Murata. Multivariate analysis for performance evaluation of active queue management mechanisms in the Internet. In *Proceedings of SPIE's International Symposium on the Convergence of Information Technologies and Communications (ITCom 2002)*, July 2002.

[39] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive RED: an algorithm for increasing the robustness of RED's active queue management," August 2001.

[40] R. Rosolen, O. Bonaventure, and G. Leduc, "A RED Discard Strategy for ATM Networks and its Performance Evaluation with TCP/IP Traffic," *ACM Comp. Commun. Review*, vol. 29, no. 3, July 1999.

[41]A. Bitorika, M. Robin, and M. Huggard, "An evaluation framework for active queue management schemes," in Proc. MASCOTS'03. IEEE, Oct. 2003,

[42] Roman A. Pletka, Andreas Kind, Marcel Waldvogel, and Soenke Mannal. Closed-loop congestion control for mixed responsive and non-responsive traffic. In Proceedings of Conference on Global Communications (GLOBECOM), volume 7, pages 4180--4186, December 2003.

[43] H. Han, C. V. Hollot, Y. Chait, and V. Misra, "TCP networks stabilized by buffer-based AQMs," in Infocom, 2004.

[44]A. Tang, J. Wang, and S. H. Low. Understanding CHOKe. In Proc. of IEEE Infocom, April 2003.

[45] Jiantao Wang, Ao Tang, and Steven H. Low. Maximum and asymptotic UDP

throughput under CHOKe. Submitted to ACM Sigmetrics, http: //netlab. caltech. edu, November 2002.

[46] R. Pan, B. Prabhakar, and K. Psounis. CHOKe: A stateless active queue management scheme for approximating fair bandwidth allocation. In Proceedings of IEEE INFOCOM 2000, pages 942--951, Tel-Aviv, Israel, April 2000.

[47]Orozco, J., Ros, D.: An Adaptive RIO (A-RIO) Queue Management Algorithm. Research Report PI-1526, IRISA (2003)

[48]Cisco: IOS Release 12.1 Quality of Service Solutions Configuration Guide—Congestion Avoidance Overview. (2002)

[49]Malouch, N., Liu, Z.: Performance Analysis of TCP with RIO Routers. Research Report RR-4469, INRIA (2002)

[50] M. Parris, K. Jeffay, and F. D. Smith, "Lightweight Active Router-Queue Management for Multimedia Networking," *Proc. Multimedia Computing and Net.*, vol. 3654, Jan. 1999, pp. 162–74.

[51] D. Gevros *et al.*, "Congestion Control Mechanisms and the Best-Effort Service Models," *IEEE Network*, vol. 15, no. 3, May/June 2001, pp. 16–26.