In compliance with the
Canadian Privacy Legislation
some supporting forms
may have been removed from
this dissertation.

While these forms may be included
in the document page count,
their removal does not represent
any loss of content from the dissertation.

Fair Bandwidth Allocation in the Internet

Gateway for Congestion Control

by

Wei Yi

B.Eng.(Electrical Engineering)
Harbin Engineering University, China

A thesis Submitted to the Faculty of Graduate Studies
and Research in partial fulfillment of the requirements for
the degree of

Master of Applied Science

Ottawa-Carleton Institute for Electrical and Computer Engineering
Faculty of Engineering

Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, K1S 5B6, Canada

September 1st, 2003
© Wei Yi

National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services

Acquisitons et
services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-88903-3

Our file Notre référence

ISBN: 0-612-88903-3

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

Canada

ABSTRACT

Currently the congestion control in the Internet is mainly achieved through end-to-end algorithms. However, many researcher had argued that such end-to-end congestion control solutions can be greatly improved when gateways have mechanisms that allocate bandwidth in a fair manner. This gateway mechanism should achieve reasonably fair bandwidth allocation and be easily implemented in high speed network. In this thesis we proposed a gateway algorithm called Virtual Round Robin (VRR). VRR allocates output bandwidth fairly among flows by applying different dropping probabilities. This algorithm uses FIFO scheduling and arriving packets are randomly dropped or placed on the queue based on flow states. The dropping decisions are simple with $O(1)$ complexity. The amount of state required to make these dropping decisions is small compared to the memory required by packets buffer. This thesis studies the performance of VRR by running the simulations using Network Simulator (NS). Both a dumbbell single congested link topology and a multiple congested links topology are used. And different types of traffic sources are considered. This simulation results show that VRR is very effective to ensure fairness and protect adaptive flows from non-adaptive flows. It outperforms the FRED and RED.

ACKNOWLEDGEMENTS

I am sincerely grateful to my thesis supervisor, Professor Ioannis Lambadaris for his guidance, encouragement, patience, and continuous support during the course of my thesis. Without his help, I could not be productive in generating the new ideas. His knowledge and effort are truly appreciated.

The best wishes to my friends, who offered their friendship and their advice and who believe in me.

Last but not least, my deepest thanks go to my family and my special yoyo, for their support and love.

TABLE OF CONTENTS

1. INTRODUCTION.....	1
1.1 BACKGROUND AND MOTIVATION	1
1.2 OBJECTIVE OF THE THESIS	4
1.3 THESIS OUTLINE	5
1.4 THESIS CONTRIBUTIONS.....	6
1.5 PUBLICATIONS	7
2. CONGESTION CONTROL MECHANISMS	8
2.1 TCP CONGESTION CONTROL ALGORITHMS	9
2.2 GATEWAY ALGORITHMS FOR FAIRNESS IN CONGESTION CONTROL.....	10
2.2.1 RED algorithm	10
2.2.1.1 Adaptive RED.....	13
2.2.1.2 Fairness problem of RED.....	15
2.2.1.3 RED with Penalty Box.....	16
2.2.2 Gateway Algorithms with per-flow Queueing and Scheduling	16
2.2.2.1 Fair Queueing (FQ).....	17
2.2.2.2 Stochastic Fair Queueing (SFQ).....	18
2.2.2.3 Deficit Round-Robin.....	21
2.2.3 Core-Stateless Fair Queueing (CSFQ)	22
2.2.4 Flow Random Early Drop (FRED)	23
2.2.4.1 Protecting low bandwidth flows	24
2.2.4.2 managing non-adaptive flows.....	25

2.2.4.3 the FRED algorithm.....	25
2.2.5 Comparison of Gateway Algorithms.....	27
3. VIRTUAL ROUND ROBIN (VRR) FOR FAIRNESS.....	29
3.1 THE PROTECTION OF FLOWS WITH LOW SENDING RATE	31
3.2 PREVENTING THE NON-ADAPTIVE FLOW FROM DOMINATING THE BUFFER SPACE.....	32
3.3 PACKET DROP ALGORITHM FOR VRR	32
3.4 ALGORITHM FOR UPDATING THE VALUE OF q_{len}	34
3.5 COMPLEXITY OF PER-FLOW STATES MANAGEMENT	35
4. SIMULATION STUDIES	38
4.1 FLOWS WITH DIFFERENT SENDING RATES	40
4.2 TCP FLOWS WITH DIFFERENT RTT.....	42
4.3 MIXTURE OF TCP FLOWS AND UDP FLOWS	45
4.4 MIXTURE OF HETEROGENEOUS FLOWS	50
4.5 MULTIPLE CONGESTED LINKS.....	52
4.6 SHORT-LIVED WEB TRANSFERS	57
5. CONCLUSIONS	62
BIBLIOGRAPHY	65

LIST OF FIGURES

Figure 1 RED Gateway Algorithm	13
Figure 2 Adaptive RED Algorithm	14
Figure 3 Fairness Queue	19
Figure 4 Stochastic Fairness Queue	19
Figure 5 Expected Collision and Variance	21
Figure 6 Updating the value of q_{len_i}	31
Figure 7 A single congested link topology	39
Figure 8 Multiple congested links topology	39
Figure 9 UDP flows with different sending rates	41
Figure 10 TCP with different RTT ($E[u]$, mean value of u)	43
Figure 11 TCP flows with different RTT (normalized u)	44
Figure 12 TCP flows with different RTT (fairness index)	45
Figure 13 One non-adaptive flow competing with multiple TCP flows under RED	47
Figure 14 One non-adaptive flow competing with multiple TCP flows under FRED	47
Figure 15 One non-adaptive flow competing with multiple TCP flows under VRR	48
Figure 16 Normalized bandwidth of a TCP flow while competing with multiple UDP flows	49
Figure 17 Fairness index when a TCP flow competing with multiple UDP flows	50
Figure 18 Mix of 9 TCP flows and 3 CBR flows	51
Figure 19 Fairness index of each 20ms period for mix traffic	52
Figure 20 Normalized throughput of a well-behaved UDP flow traversing multiple congested links	54

Figure 21 Loss rate of UDP flow traversing multiple congested links.....	55
Figure 22 Normalized throughput of a FTP TCP flow traversing multiple congested links	56
Figure 23 Loss rate of FTP TCP flow traversing multiple congested links	57
Figure 24 Cumulative fraction of web transfer delay.....	58
Figure 25 Average web transfer delay over multiple congested links.....	60
Figure 26 Loss rate of web traffic vs. number of congested links	61

Chapter 1

1. Introduction

1.1 Background and Motivation

Congestion control is one of the Internet's most fundamental architectural problems. There are two objectives for network congestion control mechanism. One is that the sources should respond to the notification of network congestion to reduce the load the network. Another objective is to allocate limited output bandwidth fairly among flows traversing the gateway, because the unfairness in allocation of bandwidth will act as an incentive for end-user to be greedy and unresponsive to network congestion notification. [1][2] Thus, the congestion control mechanism has two point of implementation. The first is at the source, where end-to-end congestion control algorithms are adopted to limit the overall network traffic when the network is congested. The second point of implementation is at the gateway. Congestion can be controlled at gateways through queueing and scheduling to determine the way in which the bandwidth is allocated to different flows at congestion, which affects the collective behavior of end-to-end control algorithms [3][4][5][6]. Therefore, gateway algorithms are crucial component in effective

congestion control. Some user may open multiple connections for one application so they can get more throughput. However, One flow can be identified by the source-destination in the gateway to prevent this.

Currently in the best effort Internet, most gateways do not actively manage per-flow bandwidth allocations [7]. In order to reduce complexity and increase efficiency, they use a single first-in first-out (FIFO) packet queue shared by all flows, and discard the incoming packets when buffer overflows. Then, the bandwidth received by a flow depends on the end-to-end congestion control algorithms used by all competing flows. The major end-to-end congestion control protocol over the Internet is the TCP [23]. The way TCP works is that it keeps increasing the sending rate of packets as long as no packet loss is detected. While the network congestion and buffer overflow, packets get dropped in the gateway. In response to that, TCP decreases the sending rate. Therefore, the transmission rate of the TCP sender depends on the level of congestion perceived in the network. To achieve relatively equitable bandwidth allocations and avoid congestion, all flows must be TCP-compatible [8][9][10]; that is, they must use a congestion control algorithm that results in bandwidth allocation similar to TCP's. However, with today's diverse and decentralized environments, it is unrealistic to expect universal implementation of any given end-to-end congestion control algorithm. Any source can obtain more bandwidth simply by using a more aggressive congestion control and can capture an arbitrarily high fraction of the bandwidth in the gateway by sending packets at a sufficiently high rate. This unfairness could become an incentive for all sources to become more aggressive or not to adopt end-to-end congestion control mechanism at all, which will threaten the entire network performance. Thus, along with the end-to-end

congestion control, gateways should play a more active role in allocating bandwidth. If the gateways ensure fair bandwidth allocations, all sources are no longer required to adhere to any particular form of end-to-end congestion control and ill-behaved sources can only have a limited negative impact on well-behaved sources. All the flow will be guaranteed a fair share in the network. If a flow send traffic more than its share, the excess packets will be dropped and it will suffer high loss rate. On the contrary, if a flow sends at a rate that is no more than the fair share, it will have very little packet losses. So the gateway will prompt the end-user to implement some end-to-end congestion control to reduce their own loss rate while achieving the fair throughput.

To accomplish the fair bandwidth allocation in the gateways, Fair queueing and related algorithms keep a separate queue for each flow and use scheduling mechanism to service each sub-queue with its fair share rate [11][12][13][14][15]. This approach can ensure max-min fair bandwidth allocation. And flows that send at less than its fair share rate will experience low packet drop rate and low queueing delay. However, the implementation complexity of this approach increases with the number of flows traversing the gateway and it does not scale well in the high-speed Internet gateway.

To reduce the complexity and increase efficiency, IP gateways usually use a single first-in first-out (FIFO) packet queue shared by all flows. They provide feedback to senders by discarding packets under overload. Random Early Detection (RED)[16][17][18] uses randomization to ensure that all connections encounter the same loss rate. The packets from the flows at high sending rate will be more likely to be dropped. However, this equal drop probability does not lead to equal bandwidth share. An ill-behaved connection

can get high fraction of the bandwidth by not reacting to packet drop and the problem of fair bandwidth allocation is not solved.

Another approach is to use per-flow information, not per-flow queueing and scheduling [19][20][21][22]. This approach uses single FIFO queue for packets from all arriving flows, but counts the number of arriving/departing packets of each active flow, and calculates its buffer occupancy. This per-flow information is used to differentiate dropping probability among connections. Normally flows with low sending rate will have less packets buffered than flows with high sending rate, so these flows will have less packet drop probability. FRED [19] is the example of this approach. This approach has considerable less complexity than the implementation of per-flow queueing and scheduling and high degree of fairness, thus provides an intermediate solution between RED and per-flow queueing. FRED requires that the gateway maintain the flow states for only active flows for which it has packets buffered. So the memory space required to maintaining per-flow states is proportional to the packet buffer size.

1.2 Objective of the Thesis

This thesis develops a mechanism, Virtual Round Robin (VRR), to fairly allocate bandwidth in the gateway. This algorithm isolates flows from each other so that aggressive flows can have a very limited impact on other flows. For those unresponsive users who send packets more than their fair share, the excess packets will be dropped and they will suffer significant levels of packet losses. This gives end users an incentive to use end-to-end congestion control because being unresponsive hurts their own performance.

There are three basic requirements of our algorithm. First, it should achieve reasonably fair bandwidth allocations. Second, for the sake of high-speed implementation, this algorithm should use FIFO packet scheduling with probabilistic drop-on-arrival. Third, this algorithm must employ some form of active queue management to respond early to congestion while absorbing small bursts of traffic. Our gateway algorithm allocates output bandwidth fairly among flows by applying different drop probability to each flow. This algorithm uses FIFO scheduling and arriving packets are randomly dropped or placed on the queue. The dropping decisions are simple with $O(1)$ complexity like RED and FRED. The amount of state required to make these dropping decisions is small compared to the memory required by packets buffer.

For evaluating the performance of the new algorithm, simulation models needed to be built. Network Simulator (NS) was selected as simulator to evaluate the new algorithm in a variety of scenarios in this thesis.

1.3 Thesis Outline

Rest of the thesis is organized as follows:

Chapter 2 gives the introduction of the congestion control mechanisms. There are two parts in the network for congestion control. For end-to-end part, TCP congestion control algorithm is briefly described. To ensure the fair bandwidth allocation, gateways play an important role in network congestion control. Various algorithms proposed are described in this chapter.

Chapter 3 gives the detail descriptions of Virtual Round Robin (VRR) gateway algorithm. This algorithm uses FIFO packet scheduling with probabilistic drop-on-arrival. The dropping probabilities of flows are differentiated based on the simple per-flow states similar to Flow Random Early Drop (FRED). The algorithms of making drop decision upon packet arrival and of flow states management are introduced in this chapter.

Chapter 4, the simulation results analysis and performance evaluation are presented. We compared the performance of VRR under a variety of scenario to two other gateway algorithms: RED and FRED.

Chapter 5 gives the conclusion of the thesis.

1.4 Thesis Contributions

The following are the main contribution of this thesis:

- Developed a gateway algorithm, Virtual Round Robin (VRR), to allocate bandwidth fairly in the network gateway. This algorithm uses a single FIFO queue for all incoming flow with probabilistic drop-on-arrival. The drop probabilities of packets from different flows are computed based on their flow states so that each flow will achieve its fair share throughput. Under VRR, flows with lower arrival rate will experience less drop probabilities.
- Implementation and validation of this Virtual Round Robin scheme in Network Simulator (NS).

- Performance evaluation of VRR through simulations. The performance of VRR was compared with other two gateway algorithms, RED and FRED. Through the simulation results, we show that both VRR and FRED can achieve much fairer bandwidth allocation than RED in all circumstance. With the same implementation complexity as FRED, VRR outperforms FRED in many cases. VRR can lessen the TCP's bias against long RTT connections and achieve fairer bandwidth allocation than FRED when TCP connections with different RTT compete for bottleneck link. VRR can also prevent non-adaptive flows from taking arbitrary portion of bandwidth at congested links. Flows that send packets at the rate higher than their fair share will have their excess packets dropped, so that well-behaved connections with a sending rate no more than their fair share will be protected.

1.5 Publications

- Wei Yi, Rupinder Makkar, Ioannis Lambadaris, Ioannis Marmorkos, "Gateway Algorithm for Fair Bandwidth Sharing", Submitted to IEEE/ICC, June 2004.
- Wei Yi, Rupinder Makkar, Ioannis Lambadaris, "Fair Bandwidth Allocation of the Gateway for Congestion Control" The 3rd IASTED International Conference on WOC, July 2003.

Chapter 2

2. Congestion Control Mechanisms

In general, there are two major objectives in the congestion control mechanism. The one is to avoid an occurrence of the network congestion, and to dissolve the congestion if the congestion cannot be avoided. The other objective is to provide fair service to connections. Up until recently, the Internet has mainly relied on the cooperative nature of TCP congestion control in order to limit packet loss and fairly share network resources. Increasingly, however, new applications are being deployed which do not use TCP congestion control and are not responsive to the congestion indications given by the network. Such applications are potentially dangerous because they drive up the packet loss rates in the network and can eventually cause congestion collapse. Without the support of network, this mechanism alone introduces several problems. The end user can intentionally or unintentionally modify the TCP code so that it does not react against the network congestion properly. Such a mis-behaving connection may continue to send packets without throttling the sending rate even if the congestion

happens in the network. If other users perform the congestion control properly, that misbehaving user may be able to receive higher throughput. This provides an incentive for end user not to adopt the end-to-end congestion control. It is becoming more critical with an increase of the commercial use of the Internet. Thus, keeping the fairness among multiple homogeneous/heterogeneous connections in the network is an essential feature for the network. A mechanism needs to be used within the Internet gateway to ensure fair resource allocation among flows traversing it. This gateway algorithm is also required to be simple enough to be implemented in a high-speed router.

2.1 TCP Congestion Control Algorithms

The TCP sender maintains congestion window ($cwnd$), and it can inject new packets into the network up to the congestion window without receipt of acknowledgements. It is flow control between TCP sender and receiver. As a congestion control mechanism, the TCP sender dynamically increase/decreases the window size according to the degree of the network congestion. The congestion level is conjectured via packet losses, which can be detected by discontinuous receipts of the acknowledgements, or timeout expiration. The timeout expiration happens if more than two or three packets in the congestion window are lost [23][24][25]. The TCP sender then recognizes that the network is congested, and throttles its window. Otherwise, it determines that the network is not congested, and inflates its window.

There are several different versions of TCP implementation. Most of the current TCP implementations are based on the TCP Reno version. TCP Reno has two phases in increasing the congestion window $cwnd$; slow start phase and congestion avoidance

phase. When an ACK packet is received by the TCP sender at time $t + t_A$, $cwnd(t + t_A)$ is updated from $cwnd(t)$ as follows;

Slow Start Phase:

$$cwnd(t + t_A) = cwnd(t) + m, \text{ if } cwnd(t) < ssth(t)$$

Congestion Avoidance Phase:

$$cwnd(t + t_A) = cwnd(t) + m^2 / cwnd(t), \text{ if } cwnd(t) \geq ssth(t)$$

Where $ssth(t)$ is the threshold value at which TCP changes its phase from the slow start phase to the congestion avoidance phase. When packet losses are detected by timeout, $cwnd(t)$ and $ssth(t)$ are updated as follows;

$$ssth(t) = cwnd(t) / 2;$$

$$cwnd(t) = m;$$

When packet losses are detected by fast retransmission algorithm, the window size $cwnd(t)$ is halved. That is,

$$ssth(t) = cwnd(t) / 2;$$

$$cwnd(t) = ssth(t);$$

TCP then enters the fast recovery phase. In this phase, the window size is increased by one packet when a duplicate ACK packet is received, and $cwnd(t)$ is restored to $ssth(t)$ when the non-duplicate ACK packet corresponding to the retransmitted packet is received.

2.2 Gateway Algorithms for Fairness in Congestion Control

2.2.1 RED algorithm

Currently most IP gateways use a single FIFO packet queue shared by all flows and use Drop Tail algorithm to discard arriving packets when the gateway's buffer space is full. Drop Tail gateways often distribute losses among connections arbitrarily [19] and tend to penalize bursty connections. The gateway algorithm RED [16] addresses Drop Tail's deficiencies. RED uses randomization to ensure that all connections encounter the same loss rate. It also try to prevent congestion, rather than just reacting to it, by dropping packets before the gateway's buffer are completely exhausted.

The RED gateway calculates the average queue size, using a low-pass filter with an exponential weighted moving average. The average queue size is compared to two thresholds, a minimum threshold ($\min_{th}$) and maximum threshold ($\max_{th}$). When the average queue size is less than the minimum threshold ($\min_{th}$), no packets are dropped. When the average queue size is greater than the maximum threshold ($\max_{th}$), every arriving packet is dropped. This forced drop ensures that the average queue size does not significantly exceed the maximum threshold ($\max_{th}$).

When the average queue size is between the minimum threshold and the maximum threshold, each arriving packet is dropped with certain probability depending on the average queue size. Each time that a packet is dropped, the probability that a packet is dropped from a particular flow is roughly proportional to that flow's share of the bandwidth at the gateway.

There are two separate algorithms in the RED gateway. The algorithm for computing the average queue size determines the congestion level in the gateway. The algorithm for calculating the packet dropping probability determines how frequently the gateway drops packets, given the current congestion level.

To calculate the average queue size, the RED gateway uses a low-pass filter. Thus, the short-term increases in the queue size that results from bursty traffic or from transient congestion do not result in a significant increase in the average queue size. The low pass filter is an exponential weighted moving average (EWMA). The weight (w_q) determines the time constant of the low-pass filter.

$$avg = (1 - w_q)avg + w_q q$$

To ensure that the gateway does not wait too long before dropping a packet and avoid dropping several consecutive packets, the RED gateway counts the number of packets that have been accepted by the gateway since the last dropped packet. The final dropping probability (P_a) is calculated based on the initial dropping probability (P_b) and the packet count ($count$) since the last dropped packet. The initial drop probability (P_b) increases linearly from 0 to maximum drop probability (max_p) as average queue size increase from minimum threshold (min_{th}) to maximum threshold (max_{th})

$$P_b = \max_p (avg - min_{th}) / (max_{th} - min_{th})$$

The final packet-dropping probability (P_a) increases slowly as the count increases since the last dropped packet.

$$P_a = P_b / (1 - count \cdot P_b)$$

The RED gateway will calculate average queue size and dropping probability at every packet arrival, and drop the packet with the final dropping probability (P_a).

Figure 1 RED Gateway Algorithm

2.2.1.1 Adaptive RED

Adaptive RED is a modified RED algorithm that was proposed in [26][17] to ensure the average queue size converge to certain target value no matter the congestion level in the gateway. The main problem of RED is that the average queue size varies with the level of congestion and with parameter settings. That is, when the link is lightly congested or $\max_p$ is high, the average queue size is near $\min_{th}$; when the link is more heavily congested or $\max_p$ is low, the average queue size is closer to, or even above, $\max_{th}$. As a result, the average queuing delay from RED is sensitive to the traffic load and to parameters, and therefore is not predictable in advance. Nevertheless, RED often does not perform well when the average queue size is close to $\min_{th}$ or $\max_{th}$. Large average queue above the $\max_{th}$ results in significantly decreased throughput and

increased dropping rates, and small average queue that close to $\min_{th}$ causes an under-utilized output link. Thus, in Adaptive RED, the parameter $\max_p$ is adjusted to keep the average queue size within a target range half way between $\min_{th}$ and $\max_{th}$. In Adaptive RED, $\max_p$ is adapted slowly, over time scale greater than a typical round-trip time, and in small steps to ensure the robustness the algorithm. An additive-increase multiplicative-decrease (AIMD) policy is used to adjust the $\max_p$.

```

Every intervalseconds:
if (avg > target and  $\max_p \leq 1$ )
increase  $\max_p$ :
 $\max_p \leftarrow \max_p + \alpha$ ;
else if (avg < target and  $\max_p \geq 0.01$ )
decrease  $\max_p$  :
 $\max_p \leftarrow \max_p \times \beta$ ;

```

Variables:
avg: average queue size

Parameters:
interval: time; 0.5 seconds
target: target for avg; $(\min_{th} + \max_{th})/2$
 α : increment; $\min(0.01, \max_p/4)$
 β : decrease factor; 0.9

Figure 2 Adaptive RED Algorithm

In the Adaptive RED, no matter the congestion level and traffic type, the average queue size is forced to oscillate around certain target value (the middle of $\min_{th}$ and $\max_p$). That means, once we set this target value, we can roughly predict the average queue size.

2.2.1.2 Fairness problem of RED

RED's goal is to drop packets from each flow in proportion to the amount of bandwidth the flow uses on the output link. It does this by dropping each arriving packet with equal probability. Therefore, the connection with the largest input rate will have the biggest drop percentage among total dropped packets. This equal dropping probability does not always lead to fair bandwidth sharing. It has bias against TCP connections with large round trip delays or traversing a large number of congested gateways. Because long RTT connections need larger windows to maintain the same throughput as shorter RTT connections and the window growth is clocked by dropping probability, the equal dropping probability of RED results in discrimination against long RTT connections. Although this is undesirable, it is not as harmful to network performance as non-adaptive flows. The non-adaptive applications don't react to the congestion in the network. When the network consists of both adaptive and non-adaptive traffic, the packets from both kind of traffic compete for output bandwidth at the gateway. As congestion builds up, packets for both types of applications may be dropped. In response to this, the adaptive applications decrease their sending rate while the non-adaptive applications may not change their sending rate of packets. As a result, the adaptive applications are penalized. Nevertheless, this sort of unfairness to the adaptive traffic acts as a disincentive for the deployment of applications incorporating end-to-end congestion control mechanisms. This drives the network to undesirable, congestion dominated operating regimes, characterized by large number of packet drops in the network with no useful work being done.

2.2.1.3 RED with Penalty Box

The RED with penalty box is proposed to prevent the damage caused by non-adaptive flows [27][28]. This approach takes advantage of the fact that high bandwidth flows see proportionally larger amounts of packet loss. By keeping a finite log of recent packet loss events, this algorithm identifies flows that are non-adaptive based on the log. Flows identified as being non-adaptive are then rate-limited. This approach is scalable and may be effective under certain scenarios. However, its performance is not clear in the face of a large number of non-adaptive flows. Unless the packet loss log is large, a single set of high bandwidth flows can potentially dominate the loss log and allow other non-adaptive flows to go through without rate-limitation. In addition, flows classified as being non-adaptive remain in the penalty box even if they subsequently become adaptive to congestion. A periodical and explicit check is thus required to move flows out of the penalty box. Finally, the algorithm relies on a TCP-friendliness check in order to determine whether or not a flow is non-adaptive. Without the knowledge of the round-trip time of every flow being multiplexed across the link, it is difficult to accurately determine whether or not a connection is TCP-friendly.

2.2.2 Gateway Algorithms with per-flow Queueing and Scheduling

The unfairness among multiple heterogeneous connections in the network will threaten the network performance and lead to congestion collapses. One solution for ensuring the perfect fairness among flows is use per-flow queueing and scheduling. This

approach can achieve good fairness among flows. However, it requires a separate FIFO queue to be maintained for each flow and it is much more complex than single-queue method such as RED. So per-flow queueing is not scalable for high-speed Internet gateway.

2.2.2.1 Fair Queueing (FQ)

Nagle [11] proposed a fair queueing (FQ) algorithm in which gateways maintain separate queues for packets from each individual flow. The queues are serviced in a round-robin manner, taking one packet from each non-empty queue in turn. Empty queues are skipped over and lose their turn. This prevents a source from arbitrarily increasing its share of the bandwidth or the delay of other sources. However, the most obvious flaw of this service is its lack of consideration of packet lengths. It provides a fair allocation of packet-sent but fails to guarantee a fair allocation of bandwidth because of variations in packet sizes. A source using long packets gets more bandwidth than one using short packets. To avoid this unfairness, Demers [13] proposed a modification of Nagle's algorithm that emulate the bit-by-bit round robin (BR) algorithm. The BR service discipline allocates bandwidth fairly since at every instant in time each flow is receiving its fair share. Let $R(t)$ denote the number of rounds made in the BR service discipline up to time t . A packet of size P whose first bit gets serviced at time t_0 will have its last bit serviced P round later, at time t such that $R(t) = R(t_0) + P$. Let t_i^α be the time that packet i belonging to flow α arrives at the gateway, and define the number S_i^α and F_i^α as the values of $R(t)$ when the packet started and finished service. With P_i^α denoting the size of the packet, the

following relations hold: $F_j^\alpha = S_i^\alpha + P_i^\alpha$ and $S_i^\alpha = \text{MAX}(F_{i-1}^\alpha, R(t_1^\alpha))$. Since $R(t)$ is a strictly monotonically increasing function whenever there are bits at the gateway, the ordering of the F_i^α values is the same as the ordering of the finishing times of the various packets in the BR discipline. BR algorithm is impractical. However, a packet-by-packet transmission algorithm can emulate the bit-by-bit round robin algorithm by letting the F_i^α define the sending order of the packets. The rule of Demers' fair queueing algorithm is that, whenever a packet finishes transmission, the next packet sent is the one with the smallest value of F_i^α . This algorithm approach the fair bandwidth allocation of the BR scheme. However, it requires $O(\log(n))$ computational complexity per packet, where n is the number of flows that are concurrently active at the gateway or router. With a large number of active flows, it is hard to implement at high speeds.

2.2.2.2 Stochastic Fair Queuing (SFQ)

SFQ was proposed by McKenney [29] to address the inefficiencies of Fair Queuing algorithm. McKenney uses hashing [30] to map packets to corresponding queues. Normally FQ would require one queue for every possible flow through the router. However, SFQ suggests that the number of queues be considerably less than the number of possible flows and uses hashing to map packets to corresponding queues. All flows that happen to hash into the same queue are treated equivalently. This simplifies the hash computation that is guaranteed to take $O(1)$ time, and allows the use of a smaller number of queues. As shown in Figure 3.1, FQ algorithm operates by maintaining a separate FCFS queue for each flow. SFQ uses a simple hash function to map from source-

destination pair into a fixed set of queues, as in the six-queue example shown in Figure 3.2. The disadvantage is that flows that collide with other flows will be treated unfairly. The fairness guarantees are probabilistic. However, if the number of queues is sufficiently larger than the number of active flows through the router, the probability of unfairness will be small.

Figure 3 Fairness Queue

Figure 4 Stochastic Fairness Queue

Refer to the analysis of [29], for a given active flow, the expected number of active flows sharing its queue is represented by

$$EC = \alpha + 1$$

and

$$VC = \frac{\alpha^2}{6} + \alpha$$

where EC is the expected number of active flows, VC is the variance, and α is the ratio of the number of active flows through the router to the number of queues. For example, if we have 100 active flows and 1000 queues, α is 0.1. A plot of these values is shown in Figure 3.3. Consider a gateway that has no flow traverse. Then a new arriving active flow is guaranteed to be given a queue with exactly one occupant (that active flow itself). This is indicated on Figure 3.3 by a value of 1 for EC and of 0 for VC when α is 0. On the other hand, when the number of active flows is as many as the number of queues (α is 1), the value of EC is 2 and VC is about 1.17. This indicates that a new active flow will share its queue with one other active flow on the average, but that the actual number of active flows may vary considerably from that value. Thus, when the occupancy α is low, the expected number and variance of active flows sharing a given queue will be low and the performance of the gateway will be consistent and predictable. Refer to [29], the number of queues need only to be a small multiple of the number of active flows.

EC-Expected Value of Collision Factor
VC-Variance of Collision Factor

Figure 5 Expected Collision and Variance

2.2.2.3 Deficit Round-Robin

The major problem of Nagle's Fair Queueing algorithm [11] is the unfairness caused by possibly different packet sizes used by different flows. Deficit Round-Robin was proposed [31] to remove this flaw, while still requiring only constant complexity. To service the queues, DRR uses round-robin servicing with a Deficit Counter assigned to each queue, which absorbs the difference of packet sizes. That is, each queue has a deficit counter, and the counter is incremented in each round, considering the pre-defined service rate. When a packet in the queue is served in the round, the counter is decremented by the amount corresponding to the packet size. If a queue was not able to send a packet in the previous round because its packet size was larger than the value of Deficit Counter, the counter value is kept and will be used in the next round. Thus,

queues that were shortchanged in a round are compensated in the next round. Through these mechanisms, DRR provides reasonable fair service among connections and it work in $O(1)$ computational complexity per packet.

While DRR reduces the computational complexity of Fair Queueing algorithm, it is still much more complex than single FIFO queue methods. Therefore, it might be difficult to apply even DRR to core routers.

2.2.3 Core-Stateless Fair Queueing (CSFQ)

CSFQ [32] is an architecture that allocates bandwidth in an approximately fair manner while allowing the routers or gateways on high-speed links to use FIFO queueing and maintain no per-flow state. This approach identifies an island of routers and distinguishes between the edge and the core of island. Edge routers compute per-flow rate estimates and label the packets passing through them by inserting these estimates into each packet header. Core routers do not maintain per flow state but instead utilize the per-flow information carried via the label in each packet's header. This label contains an estimate of the flow's rate, and then updated at each router along the path based only on aggregate information at that router. Core routers use FIFO queueing with probabilistic dropping on input. The probability of dropping a packet as it arrives to the queue is a function of the rate estimate carried in the label and of the fair share rate at that router, which is estimated based on measurements of the aggregate traffic.

In CSFQ, each flow's arrival rate $r_i(t)$ is carried in the label of each packet of that flow. Let $\alpha(t)$ be the fair share rate at time t . In general, if max-min bandwidth allocations are achieved, each flow i receives service at a rate given by $\min(r_i(t), \alpha(t))$. If $r_i(t) \leq \alpha(t)$, i.e., flow i sends no more than the server's fair share rate, all of its traffic will be forwarded. If $r_i(t) > \alpha(t)$, then a fraction $\frac{r_i(t) - \alpha(t)}{r_i(t)}$ of its packets will be dropped, so it will have an output rate of $\alpha(t)$. So each incoming packet of flow i is dropped with the probability $\max(0, 1 - \frac{\alpha(t)}{r_i(t)})$. And the arrival rate of flow i at the next hop is given by $\min(r_i(t), \alpha(t))$.

Core-Stateless Fair Queueing achieve high degrees of fairness with scalable mechanisms in the core routers, these schemes require a change in the packet format and careful configuration of routers into core, edge, and peripheral regions. Thus, while CSFQ delivers a high degree of fairness, it faces significant deployment hurdles.

2.2.4 Flow Random Early Drop (FRED)

Another approach is to use per-flow information, not per-flow queueing. This approach uses single FIFO queue for packets from all arriving flows, but counts the number of arriving/departing packets of each active flow, and calculates its buffer occupancy. This per-flow information is used to differentiate dropping probability among connections. FRED is examples of this approach.

A single FIFO queue is used for all arriving flows in the FRED. FRED allocates bandwidth among flows by enforcing different drop probabilities according to buffer occupancy of flows. FRED only keeps flow states for active flows. An active flow is the flow that has at least one packet buffered in the FIFO queue. The flow state of one flow is a count of its buffered packets. According to these flow states, FRED generate selective feedback to a filtered set of flows which have a larger number of packets queued. Thus, all the flows will have similar average buffer occupancy. In a FIFO scheduling queue, the output bandwidth distribution is proportional to buffer distribution, so managing the buffer occupancy can enforce the fairness among flows. FRED is a variant of RED except that it has per-flow state. Thus it has the parameter for RED, $\min_{th}$, $\max_{th}$, $\max_p$, w_q . Except that, FRED introduces the parameters $\min_q$ and $\max_q$, goals for the minimum and maximum number of packets each flow should be allowed to buffer. FRED also introduces the global variable $avgcq$, an estimate of average per-flow buffer count. FRED maintains a count of buffered packets $qlen_i$ for each flow that currently has any packets buffered. Finally, FRED maintains a variable $strike_i$ for each flow, which counts the number of times that the flow has failed to respond to congestion notification; FRED penalizes flows with high strike value.

2.2.4.1 Protecting low bandwidth flows

Low bandwidth flows usually have less backlogged packets and should experience low packet drop rate. FRED allows each flow to buffer $\min_q$ packets without loss. All additional packets are subject to RED's random drop. Thus an incoming packet is always

accepted if the flow has fewer than $\min_q$ packets buffered and the average buffer size is less than $\max_{th}$. Therefore the low bandwidth flow will have lower drop probability. The value of $\min_q$ is not less than 4. If the value of $avgccq$ is greater than 4, the value of $\min_q$ is set to $avgccq$.

2.2.4.2 managing non-adaptive flows

An adaptive flow can adjust its sending rate to whatever bandwidth the network provides. When the network is congested, it will reduce its sending rate. A TCP sender exhibits this property. A non-adaptive flow, in contrast, can consume a large portion of the gateway buffers by injecting more packet arrivals than departures. In the FIFO scheduling, the output bandwidth distribution equals the buffer occupancy distribution. Thus flows with more buffered packets will have more bandwidth. FRED never lets a flow buffer more than $\max_q$ packets, and counts the number of times each flow tries to exceed $\max_q$ in the per-flow $strike$ variable. Flows with high $strike$ values are not allowed to buffer more packets than the average flow. This allows adaptive flows to send bursts of packets, but prevents non-adaptive flows from consistently monopolizing the buffer space.

2.2.4.3 the FRED algorithm

Parameters:

$\min_{th}$: minimum threshold of RED

$\max_{th}$: maximum threshold of RED

$\max p$: maximum dropping probability of RED

w_q : the weight of EWMA

$\min_q = 2$;

Global Variables:

$qlen$: total queue size

avg : the average of $qlen$

$avgcq$: average per-flow queue size

$\max_q$: maximum allowed per-flow queue size;

N_{active} : no. of active flows

Per-flow Variables:

$qlen_i$: number of packets buffered of flow i

$strike_i$: number of over-runs;

For each arriving packet from flow i :

Calculate the avg ;

Compute the drop probability P based on RED algorithm;

$avgcq = avg / N_{active}$;

$\max_q = \min_{th}$;

if ($avg \geq \max_{th}$) $\max_q = 2$;

//identify and manage non-adaptive flows

if ($qlen_i \geq \max_q \parallel (avg \geq \max_{th} \&\& qlen_i > 2 * avgcq) \parallel (qlen_i \geq avgcq \&\& strike_i > 1)$) {

```

        strike++;

        drop the packet;
    }

    else if (qleni ≤ MAX(minq, avgq) && (avg ≤ maxq)

        Accept the packet

    else //operating in random drop mode

        drop the packet with the probability P;

```

2.2.5 Comparison of Gateway Algorithms

Among the algorithms mentioned above, RED gateway requires least complexity, but its mechanism is not adequate for ensuring fair allocation. Router mechanism like fair queuing can achieve very fair bandwidth allocation and have many desirable properties for congestion control in the Internet. However, the complexity of per-flow queuing and scheduling may prevent them from being cost-effectively implemented. While Core-Stateless Fair Queueing (CSFQ) provides an elegant and efficient solution to providing fairness, it relies on the use of additional information that is carried in every packet of the flow and careful configuration of edge routers and core routers. It is difficult to be deployed. Thus, the approach that relies on FCFS scheduling and per-flow buffer occupancy management becomes an appropriate intermediate solution. The idea behind FRED is to keep state based on instantaneous queue occupancy of a given flow. The flows that occupy a larger portion of buffer space will have higher drop probability. In

the following chapters, we will present another gateway algorithm, Virtual Round-Robin, that allocate bandwidth fairly through per-flow states. It has the similar complexity as FRED and achieves higher degree of fairness as shown in our simulation studies.

Chapter 3

3. Virtual Round Robin (VRR) for fairness

We present a gateway algorithm VRR (Virtual Round Robin) in this chapter. VRR is a modification of RED that uses per-flow states to differentiate the dropping rate of individual flow. For each flow, VRR maintains two per-flow variables, q_{len_i} and $strike_i$. q_{len_i} represents the virtual buffer occupancy of flow i and $strike_i$ indicates whether flow i is responsive to congestion. The per-flow state q_{len_i} s of VRR are not the actual buffer occupancy of each flow, but the number of buffered packets as if all flows were serviced in fair round robin fashion. All packets accepted will be enqueued to single FIFO queue. The value of q_{len_i} is decrease in a round robin fashion, and is increased when the packets from flow i are accepted. Thus, the value of q_{len_i} represents the size of sub-queue i as if all the flows have their own sub-queue and are serviced in round-robin fashion. Although all the packets are actually serviced in single FIFO queue, the drop probability of flow i is calculated based on the value of q_{len_i} . Thus, this gateway will have the same drop behavior as the gateway with per-flow queueing and scheduling.

Since all the packets accepted will be eventually transmitted, the bandwidth distribution is actually determined by how the packets get dropped. The amount of packets accepted from one flow during certain time interval is roughly equal to the amount of packets serviced from that flow. if number of flows in the gateway is N , the total queue length $qlen = \sum_{i=1}^N qlen_i$. If the link capacity is B , then after a period of time T the value of $qlen$ should be decreased $B \times T$. Because all the $qlen_i$ s are decreased in a round robin way, they will be decreased equally of value $B \times T / N$. So the rate that each $qlen_i$ is decreased is B/N , which is the fair share rate of flow i . As shown in Figure 1, the rate that packets accepted from one flow cannot be more than its fair share rate, otherwise the value of $qlen_i$ will grow to infinity. Thus, the service rate of each flow should be no more than its fair share in long run. This algorithm can achieve similar performance as gateway algorithms with per-flow scheduling in bandwidth allocation, but with significantly less complexity.

1. $\mu(t) = B / N(t)$, where B is the link capacity, and $N(t)$ is the number of active flows
2. $\lambda_i(t) \leq \mu(t)$, otherwise the value of q_{len_i} will grow to infinity.

Figure 6 Updating the value of q_{len_i}

3.1 The protection of flows with low sending rate

VRR drops packets just like RED, except that each flow can increase its q_{len_i} up to a minimum value min_q without experiencing packet loss. If the q_{len_i} of one flow exceeds the min_q , packets from that flow will be dropped with the probability calculated through RED algorithm. The value of q_{len_i} is increased when a packet from flow i is accepted and decreased at the fair share because all q_{len_i} s are decrease in round robin way. When the arriving rate of a flow is less than that rate, q_{len_i} will be less than min_q , so the packets from that flow will be always accepted without loss.

3.2 preventing the non-adaptive flow from dominating the buffer space

In VRR, the rate that packets of flow i is accepted is always no more than the fair share rate that q_{len_i} is decreased at. Otherwise, the value of q_{len_i} will grow to infinity, which is impossible because all the arriving packets will be dropped if the value of q_{len_i} keeps growing.. Thus, all the active flows should get no more than their fair bandwidth share. However, for a non-adaptive flow, the value of its q_{len_i} may grow to a large value because it does not reduce their sending rate. If there are large number of non-adaptive flows, the buffer will be dominated by them and become full. In this case, all incoming packets get dropped including the packets from well-behaved flows. To solve this problem, the non-adaptive flow has to be identified and its q_{len_i} is not allowed to exceed a limit. In VRR, we maintain another flow state $strike_i$ for each active flow. When the buffer becomes too full, the flow with largest q_{len_i} is identified as non-adaptive and its $strike_i$ is set to 1. The flow recognized as non-adaptive is not allowed to have its q_{len_i} higher than min_q and all the incoming packet from that flow will be dropped if the q_{len_i} exceed that limit. This will prevent the flows with persistent high sending rate dominating the buffer.

3.3 Packet Drop algorithm for VRR

VRR drops packets just like RED, except that each flow can increase its virtual queue size up to a minimum value min_q without experiencing packet loss. If the virtual queue

size of one flow exceeds the $\min_q$, packets from that flow will be dropped with the probability calculated through RED algorithm. To manage the unresponsive flow, VRR maintains a variable $\min_q$ for each flow. Whenever the queue becomes too full, the flow with largest virtual queue size will be deemed as unresponsive and its strike_i will be set to 1. Thus, all the arriving packets for that flow will be dropped if its virtual queue size is greater than $\min_q$. If virtual queue of flow i becomes 0, strike_i is set back to 0. The drop algorithm for VRR is as follow:

Parameters:

$\min_{th}$: minimum threshold of RED

$\max_{th}$: maximum threshold of RED

$\max p$: maximum dropping probability of RED

w_q : the weight of EWMA (exponential weighted moving average)

q_{lim} : Buffer size

$\min_q$: minimum value of each virtual queue size, set to 4 in our simulations

Global Variables:

q_{len} : total queue size

avg : the average of q_{len}

$\max Flow$: index of the virtual queue that have the most buffered packets

N_{act} : No. of active flow

Per-flow Variables:

q_{len_i} : virtual queue size of flow i

`strike`: if the value is 1, flow i can have no more than min_q packets buffered

For each arriving packet from flow i :

Calculate the `avg`;

Compute the drop probability P based on RED algorithm;

//when buffer overflow, mark the flow with most packets buffered as unresponsive

if ($\text{avg} \geq \text{max}_{th}$ || $q_{len} \geq q_{lim}$)

$\text{strike}_{\text{maxFlow}} = 1$;

if ($q_{len} \leq \text{MAX}(\text{min_q}, \frac{\text{min}_{th}}{N_{act}})$)

Accept the packet;

//unresponsive flow is not allowed to buffer more than $\frac{\text{min}_{th}}{N_{act}}$ packets

else if ($\text{strike} == 1$)

drop the packet;

else //operating in random drop mode

drop the packet with the probability P ;

3.4 Algorithm for updating the value of q_{len}

In VRR we decrement the value of q_{len} s in a round-robin way when the packets are serviced, so that the q_{len} will be the supposed number of buffered packets of flow i if the packets from each flow are dequeued in a fair round-robin way. We do this by keeping an `ActiveList`, which is a list of indices of non-zero q_{len} . The algorithm is:

Whenever a packet from flow i is accepted and enqueued:

$qlen_i++$;

if ($qlen_i==1$)

 append i to the end of ActiveList;

whenever a packet dequeued :

let k be the first element of ActiveList;

$qlen_k--$;

remove the k from ActiveList

if ($qlen_k>0$)

 append k to the end of ActiveList;

else discard k ;

3.5 Complexity of per-flow states Management

VRR uses flow states instead of separate FIFO queue for each flow. This greatly reduces the complexity of per-flow queueing and scheduling, because each flow state can be as simple as a piece of memory to hold an integer variable. For maintaining these per-flow states, we use an array of integer pair to hold the value of $qlen_i$ and $strike_i$ of flows. , If we use the source-destination pair of IP address to identify the flow, the number of

possible flow will be 2^{64} . Then it is infeasible to keep per-flow state for every possible flow, because the size of the array need to be 2^{64} .

In this thesis we use an approach similar to SFQ to manage flow states, which doesn't need to keep flow state for every possible flow. In this approach, the size of array of flow states should be relative large compared to number of active flows, but a lot less than the number of possible flows. A simple hash function is used to map from source-destination pair into one of index of array. Since the number of indexes of array is less than the number of possible source-destination pair, it is possible that more than one active flow will be mapped into the same index. If two active flows happen to be mapped into the same index, we call it collision. When a collision happens, two flows will share the same flow state and receive less than their share of bandwidth. However, because the number of indexes is large compared to the number of active flows, each active flow will have high probability not to collide with other active flows. It is a stochastic assurance that each flow is assigned to its own flow state like Stochastic Fair Queueing (SFQ).

The hash function we used is:

$$\text{Index} = ((\text{src_addr} \ll 1) + \text{dst_addr}) \bmod P$$

Where P is chosen to be a prime and is the size of the flow state array. The shift is needed in order to map a TCP connection into two flows, one for each direction. After the hash calculation, the flow states are accessed from the memory with $O(1)$ time complexity.

Contrasting with a separate FCFS sub-queue, a flow state is only a piece of memory that holds the information about an active flow. In VRR, a flow state contains 2

integer variables, $qlen_i$ and $strike_i$, that need 8 bytes. As suggested in [29], the size of flow state array should be more than 10 times the number of active flows. Because the number of active flows cannot be more than the size of packet queue in the gateway, the memory space needed is:

$$memory_size \geq 40 \times qsize$$

where $memory_size$ is the memory required, $qsize$ is the number of packets allowed in the queue. For example, if the queue size is 100 packets, the memory required should be 8Kbytes, which is equivalent to size of 16 average IP packets. Thus, only 16% of extra memory is required in VRR to store flow states in addition to the memory needed for packet buffering. On the other hand, the more memory we use, the more flow states we can have, and the less chance a collision will happen.

Chapter 4

4. Simulation Studies

We evaluate the performance of VRR by simulations in NS [33]. As a benchmark, we compare the performance of this gateway algorithm to RED and FRED. All data packets in our simulations are 1000 bytes. To set the parameters of RED, we follow the guideline from [34]. The $\min_{th}$ is set to 30 packets and $\max_{th}$ is three times $\min_{th}$. The w_q is set to 0.002 and the value of $\max_p$ is set to 0.1. All the RED gateways have the buffer limit of 200 packets. The VRR and FRED have the same parameters as in the RED setting above, except that the $\min_q$ of FRED and $\min_q$ of VRR are 4 as recommended in [19].

Figure 7 A single congested link topology

Figure 8 Multiple congested links topology

Both a dumbbell single congested link topology (Figure 7) and a multiple congested links topology (Figure 8) are used. To evaluate the performance, we measure the average throughput of each flow in all our simulations. In some cases, we also calculate the Jain's fairness index [35] to measure how effective different gateway algorithms can fairly allocate the bandwidth among flow. The fairness index F is define as:

$$F = \frac{(\sum_{k=0}^{N-1} b_k)^2}{N \sum_{k=0}^{N-1} b_k^2} \quad (1)$$

Where b_i is the throughput of flow _{i} and N is the total number of active flows in the gateway.

4.1 Flows with different sending rates

In this experiment, we test the performance of gateway algorithms when flows with different arrival rate compete at the bottleneck link as in the topology in Figure 7. Under an ideal algorithm, the bandwidth achieved by each flow should be no more than its fair share rate no matter its sending rate. The excess packets from the flow that has higher arrival rate than its fair share will be dropped in the gateway and the flows with a sending rate less than their fair share will experience no packet drop. In this experiment, we have 10 UDP flows, indexed from 0 to 9. The sending rate of the first flow is the 1Mbps, which is equal to fair share rate, and every subsequent flow sends at a rate 0.2Mbps higher than the previous flow. Thus flow 0 transmits at 1.0 Mbps, flow 1 transmits at 1.2 Mbps, and the last flow transmits at 2.8 Mbps.

Figure 9 UDP flows with different sending rates

Figure 9 summarizes the bandwidth share of each flow gotten when different gateway algorithms are used in the bottleneck link. VRR has the best performance in this scenario. Each flow got exactly its fair share and well-behaved flow0 suffer no packet loss because it sends packets at its fair share rate. On the contrary, bandwidth share of each flow is proportional to its sending rate with RED. FRED is much fairer than RED, all the flows receive roughly their fair share. However, the throughputs of high sending rate flows are still slightly higher than low sending rate flows. The throughput of flow0 is 0.94Mbps in FRED gateway. That means flow0 suffered 6% packet drop rate even though it sends at its fair share, and the flow with high sending rate will achieve higher throughput.

4.2 TCP flows with different RTT

In this experiment we show the effectiveness of gateway algorithms to protect the TCP flow with large round-trip delay. TCP flows with large RTT will get less throughput than its fair rate when competing with TCP flows with short RTT. This bias against connections with large round-trip delays is because these connections need large windows to maintain the same throughput as shorter round-trip time connections. However, since window growth is clocked by returning acknowledgements (ACKs), and the window size is increased every RTT, large RTT connections that need large windows actually have their windows grow slower. This results in discrimination against long RTT connections. In this simulation we let n TCP flows traverse the gateway, where flow0 has long RTT(100ms) and the other flows have short RTT(4ms). Because the TCP's bias against the large RTT flows, flow0 normally get less throughput compared to the other flows. We measure u , the difference between long RTT TCP bandwidth and average low RTT TCP bandwidth, where

$$u = \frac{1}{n} \sum_{i=1}^{n-1} B_i - B_0 \quad (2)$$

The n is the number of TCP flows, and B_i is the bandwidth of flow _{i} . In an ideal case, the flow0 with less throughput has low drop probability and its congestion window can grow bigger than that of flows with short RTT. Thus, the throughput of flow0 increases and the value of u should be small.

Under VRR and FRED, each simulation is repeated 20 times and the mean value of u , $E[u]$, are computed.

Figure 10 TCPs with different RTT ($E[u]$, mean value of u)

As shown in the Figure 10, the mean value of u in VRR is always smaller than in FRED. However, flow0 still gets less throughput than other flows under VRR even though VRR adopt a fair round-robin fashion to decrease q_{len_i} so that each flow that has non-zero value of q_{len_i} should obtain its exact fair share. This is because the TCP flow's sending rate is fluctuating while the size of congestion window is changing. For certain period of time, its sending rate is less than its fair share and q_{len_i} becomes zero. When q_{len_i} becomes zero, the value of q_{len_i} cannot be decreased and flow_i will miss its round. As the point of view of gateway, this flow temporarily becomes inactive and cannot claim its share of bandwidth. For all the active flows, the fairness is still maintained. In another

plot, we consider the ratio between the value of u and the fair share throughput in each simulation, $u / \text{fairShare}$. This plot is shown in Figure 11.

Figure 11 TCP flows with different RTT (normalized u)

In the case of one 4ms flow0 competing with 4 100ms flows, the value of u is 33% of each flow's fair share rate in FRED, and under VRR this percentage is only 10.1%.

The Jain's fairness index F shown in equation (1) is also plotted. As shown in Figure 12, the VRR is fairer than FRED in all cases.

Figure 12 TCP flows with different RTT (fairness index)

4.3 Mixture of TCP flows and UDP flows

In the first experiment we show the impact of one non-adaptive UDP flow on a set of adaptive TCP flows. In this simulation, there are 5 flows traverse the bottleneck link. We assume that flow 0 is running non-adaptive UDP while the rest of the flows are adaptive TCP connections. The propagation delay of bottleneck link is 1ms. The propagation delay of access links is 5ms. Each simulation is run for 200s and the latter half of the simulation is used for the purpose of bandwidth calculation. The simulation results are shown in Figure 13-15. As we can see in Figure 13, RED has no control over non-adaptive flow when it competes to adaptive TCP flows. The higher sending rate the CBR flow has, the more bottleneck bandwidth it can obtain. The other TCP flows can only share the leftover

bandwidth. With the growth of CBR sending rate to the link rate, TCP flows get very little share.

Comparing to RED, other active queue management schemes in our experiment can all effectively prevent the non-adaptive flow from taking all the bottleneck bandwidth. In VRR, the CBR flow may get higher bandwidth than TCP flows when its sending rate is higher than 2Mbps as seen in Figure 15. This is because CBR flow is always active in the gateway when its sending rate is higher than fair share and TCP flows will become temporarily inactive because of the fluctuation of their sending rate. Although the fairness is still ensured for all the active flows in VRR, the bandwidth of non-adaptive flow is still higher than other adaptive flows. However, comparing to Figure 14, the performance of VRR is still better than FRED in this scenario. Under VRR flow0 can get no more than 2.3Mbps throughput, which is 15% more than its fair share. Under FRED flow0 will achieve 2.9Mbps throughput, which is 45% more than its fair share.

Figure 13 One non-adaptive flow competing with multiple TCP flows under RED

Figure 14 One non-adaptive flow competing with multiple TCP flows under FRED

Figure 15 One non-adaptive flow competing with multiple TCP flows under VRR

In the following experiment, we measure how well the algorithms can protect a single TCP flow against multiple non-adaptive UDP flows. We consider experiments involving N flows, $N = 2 \dots 10$. Out of these we assume $N - 1$ UDP flows and one TCP flow for each experiment. Each UDP sends at twice its fair share rate of $\frac{10}{N}$ Mbps. Figure 16 plots the ratio between the average bandwidth of the TCP flow and the fair share bandwidth it should receive. VRR has the best performance across the entire range; TCP flow can achieve its ideal fair share. In RED, TCP flow is completely shut out by the non-adaptive flows and obtains no bottleneck bandwidth. FRED can also protect the TCP flow effectively from being shut out. However, it is harder for TCP flow to get its fair share under FRED as the number of competing UDP flows increases. In case of 9 competing

UDP flows, the TCP flow only achieves 46.4% of its fair share. The Jain's fairness index [35] was calculated in all cases. As shown in Figure 17, the value of fairness index under FRED get lower with the number of total flows, while the fairness index under VRR remain close to 1 all the time.

Figure 16 Normalized bandwidth of a TCP flow while competing with multiple UDP flows

Figure 17 Fairness index when a TCP flow competing with multiple UDP flows

4.4 Mixture of heterogeneous flows

This experiment has a mix of TCP and CBR UDP flows. There are 9 TCP flows and 3 CBR flows. The TCP flows have different round-trip times; the first 3 TCP have round-trip times close to 20ms, the next three have RTTs around 40ms, and the last three have RTTs around 60ms. The CBR flows, with flow numbers 9, 10, 11, have sending rate of 5 Mbps, 3 Mbps and 1Mbps respectively. Figure 18 shows the bandwidth of each of the 12 flows with RED, with FRED and with VRR. With RED, the high-bandwidth CBR flows get almost all the bandwidth, leaving little for the TCP flows. All the flows experience the same loss rate, so the flows with higher sending rate will achieve higher bandwidth. As seen in Figure 18, under RED flow9 gets 4.7Mbps bandwidth which is close to its 5Mbps sending rate. Both VRR and FRED are able to restrict the bandwidth received by

the CBR flows to near the fair share. However, with VRR all the flows have a more evenly distributed bandwidth share.

Figure 18 Mix of 9 TCP flows and 3 CBR flows

In a subsequent simulation, we compare the performance when the traffic rates are randomly changing. We use the same traffic combination as above. We assume intervals of 20 seconds each. During each interval a flow has 60% probability to transmit. We run FRED and VRR as gateway algorithms and compute the fairness index [35] of each 20s time interval.

Figure 19 Fairness index of each 20ms period for mix traffic

As shown in figure above, the fairness index of VRR is higher than that of FRED all the time. Also the value of fairness index under VRR is much smoother than the one under FRED.

The average value of fairness index of VRR is 0.986, and confidence interval is [0.981, 0.991]

The average value of fairness index of FRED is 0.848, and confidence interval is [0.822, 0.874]

4.5 Multiple congested links

We now analyze how the throughput of a well-behaved flow is affected under different gateway algorithm when the flow traverses more than one congested link. The simulations are performed based on the topology shown in Figure 8. Except the flow from the sender to receiver, all flows are 2Mbps UDP flows. The capacity of each link in the system is 10 Mbps, and each link between gateways has 9 UDP flows at 2 Mbps sending rate. This will cause all links between gateways being congested.

In the first simulation, we let a UDP flow sending at its fair share rate of 1Mbps from Sender to Receiver. In an ideal case, this flow should suffer no packet loss and have all its traffic forwarded because it transmits at its fair share. Figure 20 shows the fraction of UDP flow that is transferred versus the number of congested links. VRR has the best performance in this case. With the increase of the number of congested links, the throughput of well-behaved UDP flow is always close to its fair share bandwidth. FRED also performs significantly better than RED. The VRR outperform the FRED in the entire range in Figure 20. When the number of congested links is 10, the well-behaved UDP flow under FRED achieves only 82% of its fair share, while under VRR it obtains its full fair share. There is an 18% difference in bandwidth sharing between FRED and VRR in this case.

Figure 20 Normalized throughput of a well-behaved UDP flow traversing multiple congested links

Figure 21 plot the loss rate of that well-behaved flow under different algorithms. Under VRR, this well-behaved flow experience very little packet loss compared to FRED. Under RED, the test flow will experience high fraction of packet loss at each congested link, so the loss rate is very high and increases with the number of links.

Figure 21 Loss rate of UDP flow traversing multiple congested links

In another simulation, we show the behavior of TCP flow traversing multiple congested links. We use the same topology as in Figure 8, and send a FTP flow from sender to receiver instead of a UDP flow. In an ideal gateway, this test flow should experience very little packet loss at every congested link when its arrival rate is lower than its 1Mbps fair share. And this low drop rate will allow the TCP congestion window of this connection grow, so this flow can achieve its 1Mbps fair share throughput. The actual throughput of test TCP flow under different algorithms is shown in Figure 22, when TCP flow traverses multiple congested links, its throughput decreases in all gateway algorithms with the number of congested links because its packets are more likely to be dropped. As an extreme case, TCP flow gets completely shut down when the congested gateway uses RED. Under FRED, the throughput the TCP flow decreases significantly with the number

of congested links increase. VRR outperforms FRED over all range. Under VRR, flow0 achieve 78.3% of its fair share when flow0 traverse 10 congested links, it achieves 78.3% of its fair share under VRR and only 5.9% under FRED.

Figure 22 Normalized throughput of a FTP TCP flow traversing multiple congested links

We also plot the loss rate of flow0 under FRED and VRR in Figure 23. When the number of congested links is high, RED drops all the packets from flow0, and flow0 experiences small loss rate under FRED and VRR. The loss rate under VRR is lower than FRED so TCP flow can achieve higher throughput. As the number of congested links increases, the loss rate under VRR decreases because the RTT of TCP flow0 grows. With the same loss rate, large RTT TCP connections will achieve less throughputs. Under VRR, the TCP

flow0 with less throughput has low drop probability and its congestion window can grow bigger to allow its throughput increase. Thus, the loss rate of flow0 under VRR decreases as the number of links increases. Under FRED, the TCP flow0 experiences packet losses at each congested link and the loss rate increases with the number of the links. Both the higher loss rate and large RTT contribute to the significant decrease of flow0's throughput under FRED.

Figure 23 Loss rate of FTP TCP flow traversing multiple congested links

4.6 Short-lived web transfers

In this simulation we assess the performance of VRR with short-lived web traffic. Short-lived web flows spend significant amount of time during the slow start phase, so they send at low rate most of the time. A fair gateway algorithm should be able to reduce the loss rate of web traffic and allow the short flow to finish sooner. The background traffic

in the simulation is the mix of long lived FTP flows and unresponsive UDP flows. There are 9 FTP flows with different RTT and 3 CBR flows with 5Mbps, 3Mbps, and 1Mbps sending rate respectively. In addition, we consider 600 web-objects transfer with average of 20 packets per web-object to transfer [36][37]. We use the built-in web-traffic model in NS [33]. The competing flows starts randomly within the initial 5s of simulation while the web-traffic start after 50s. We record the object transfer delays for each web transfer. Figure 24 is the cumulative distribution of transfer delay of web-traffic under different gateway algorithms.

Figure 24 Cumulative fraction of web transfer delay

Both FRED and VRR can reduce the loss rate of web traffic compared to RED. So the transfer delay is smaller in these two algorithms. The results are summarized in Table below:

Algorithm	Transfer delay	Std. dev	Loss rate
RED	2.1123	2.8882	5.36%
FRED	0.6344	0.3997	0.84%
VRR	0.5976	0.3011	0.52%

Because the loss rate in VRR is less than in FRED, the transfer delay under VRR is less than under FRED.

As the experiment above shown, both FRED and VRR can protect low bandwidth web-transfer flow. Therefore these two algorithms can reduce the transfer delay of short web traffic. We now study the performance of these algorithms when the web connection traverse multiple congested link. We use the topology of Figure 8. Each congested link has 10Mbps capacity and 10 FTP background connections. The short web-traffic connections start after 50s simulation time from the sender to receive. Each simulation is repeated 5 times and the mean loss rate and transfer delay are the average of all simulations. We record the average object transfer delay for web transfers in figure 25. As shown in the figure 25, as the number of congested links increase, the mean transfer delays are increasing. This is because the packets from web-traffic are more likely to be dropped when they go through multiple congested links.

Figure 25 Average web transfer delay over multiple congested links

The figure 26 shows that the loss rates increase with the number of congested links. Also, when the packets traverse multiple congested links, they will experience higher queueing delay and propagation delay. As seen in figure 26, the loss rate of web packets in VRR does not increase with the number of links, and the average transfer delays of web traffic are smaller than FRED in all range of figure 25. The transfer delays in VRR increase with no. of links only because the queueing delay and propagation delay are increasing. As the round-trip delay increases, the web-transfers have lower throughput and thus experience less loss rate under VRR. On the contrary, under FRED the web-transfers have higher packet loss rate with the increase of number of links, even though the throughputs are lower. This shows that VRR has better protection of low bandwidth flows than FRED.

Figure 26 Loss rate of web traffic vs. number of congested links

Chapter 5

5. Conclusions

Fair bandwidth allocation among flows in the gateway is important for Internet congestion control. We propose a Virtual Round Robin (VRR) gateway algorithm to enforce per-flow fair bandwidth allocation by keeping per-flow information. This mechanism achieves reasonably fair bandwidth allocation and is easily amenable to high-speed implementations. It uses a single FIFO queue with probabilistic drop-on-arrival; when a packet arrives either the packet is dropped or placed on a FIFO queue. The drop probabilities of packets from different flows are computed based on their flow states so that each flow will achieve its fair share throughput. Under VRR, flows with lower arrival rate will experience less drop probabilities. The dropping decisions are simple with $O(1)$ complexity (that is, the complexity doesn't increase with the number of flows or packets). A VRR gateway maintains state information only for active flows, so the amount of storage required is small.

In our simulation study, the performance of VRR was compared with other two gateway algorithms, RED and FRED. Through the simulation results, we show that both VRR and FRED can achieve much fairer bandwidth allocation than RED in all circumstance. With

the similar implementation complexity as FRED except for the maintenance of an extra ActiveList, VRR outperforms FRED in many cases.

- When UDP flows with different sending rate traverse the same congested link, under RED all the flows experience the same packet loss rate, so the bandwidth of each flow is proportional to its sending rate. Under VRR all the flows get the same bandwidth share. VRR can allocate bandwidth more fairly than FRED in this case.
- TCP has bias against long RTT connections. Long RTT TCP connections will get less throughput than short RTT TCP connections when they are competing for one bottleneck link. VRR can help the long RTT TCP connections to get higher throughput, so that the bottleneck link bandwidth will be shared fairly. Under VRR the differences between the throughputs of long RTT connections and short RTT connections are smaller than under FRED.
- When one non-adaptive flow competes with TCP flows, as the non-adaptive flow increases its sending rate, both FRED and VRR can prevent it from taking all the bottleneck bandwidth. However, under VRR, this non-adaptive flow can grab less excess bandwidth than FRED.
- In the case of one TCP flow competing with multiple non-adaptive UDP flows, VRR can protect this adaptive flow to get its fair share while it get much less bandwidth under FRED.
- A well-behaved flow will be protected under VRR even though it traverses multiple congested links. Under FRED, the well-behaved flow suffer higher

packet loss rate and get less bandwidth when the number of congested links increases.

- In the short-lived web transfer, both VRR and FRED are able to reduce the loss rate of web traffic and allow the short flow to finish sooner than RED. When the web transfers are traversing multiple congested links, VRR performs better than FRED.

Under VRR, the flows with lower arrival rate experience lower loss rate. Flows that send packets at the rate higher than their fair share will have their excess packets dropped, so that well-behaved connections with a sending rate no more than their fair share will be protected. It can prevent non-adaptive flows from taking arbitrary portion of bandwidth at congested links and lessen the TCP's bias against long RTT connections.

Bibliography

- [1] P. Gevros, J. Crowcroft, P. Kirstein, S. Bhatti, "Congestion Control Mechanisms and the Best Effort Service Model", IEEE Network, May/June 2001.
- [2] G. Hasegawa, M. Murata, "Survey on Fairness Issues in TCP Congestion Control Mechanisms", IEICE Trans. Commun., Vol.E84-B, No.6 June 2001.
- [3] A. Mank, K. Ramakrishn, "Gateway Congestion Control Survey", RFC1254, IETF, August 1991.
- [4] B. Suter, T.V. Lakshman, D. Stiliadis, A. Choudhury, "Design Considerations for Supporting TCP with per-flow Queueing", In Proc. INFOCOM, April 1998.
- [5] A. Legout, E. W. Biersack, "Revisiting the Fair Queueing Paradigm for End-to-End Congestion Control", IEEE Network Magazine, Feb. 2002.
- [6] E. Hahne, "Round-Robin Scheduling for Max-Min Fairness in Data Networks", IEEE Journal of Selected Areas in Communications, 1991.
- [7] B. Braden, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, IETF, April 1998
- [8] S. Floyd, M. Handley, J. Padhye, J. Widmer, "Equation-Based Congestion Control for Unicast Applications", SIGCOMM 2000.
- [9] S. McCanne, V. Jacobson, "Receiver-driven Layered Multicast", in Proceeding of ACM SIGCOMM, pp. 117-130, Sept. 1996.
- [10] J. Widmer, R. Denda, M. Mauve, "A Survey on TCP-Friendly Congestion Control", IEEE Network, 2001.

-
- [11] J. Nagle, "On packet switches with infinite storage", Internet Engineering Task Force, RFC-970, December, 1985.
- [12] J.R. Davin and A. T. Heybey. A simulation study of fair queueing and policy enforcement Computer Communication Review, 20 5 23-29, October 1990
- [13] A. Demers, S. Keshav, S. Shenker, "Analysis and simulation of a fair queueing algorithm", Proceedings of ACM SIGCOMM'89, pp 3-12.
- [14] E. Hashem, R. Gallager, "Round Robin Scheduling for Fair Flow Control in Data Communications Networks", IEEE International Conference on Communications, June 1986.
- [15] D. Stiliadis, A. Varma, "Efficient Fair Queueing Algorithms for Packet-Switched Network", in IEEE/ACM Transactions on Networking, April 1998, pp. 175-185.
- [16] S. Floyd, V. Jacobson, "Random Early Detection for Congestion Avoidance", IEEE/ACM Transactions on Networking, August 1993.
- [17] Sally Floyd, R. Gummadi, and S. Shenker, "Adaptive RED An Algorithm for Increasing the Robustness of RED's Active Queue Management", Available at <http://www.icir.org/floyd>, August 1, 2001.
- [18] B. Braden, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, IETF, April 1998.
- [19] D. Lin, R. Morris, "Dynamics of Random Early Detection", in Proc. of SIGCOMM, 1997.
- [20] F. Anjum, L. Tassiulas, "Fair Bandwidth Sharing among Adaptive and Non-Adaptive Flows in the Internet." Proc. IEEE INFOCOM '99, Mar. 1999.

-
- [21] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker, "Approximate Fairness through Differential Dropping", 2001. Work in progress.
- [22] R. Mahajan, S. Floyd, D. Wetherall, "Controlling High-Bandwidth Flows at the Congested Routed", Technical report tr-01-001, ICSI, April 2001.
- [23] V. Jacobson, "Congestion Avoidance and Control", In Proceeding of SIGCOMM 88, Stanford, CA, August 1988.
- [24] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery Algorithms", RFC 2001, Jan 1997.
- [25] M. Allman, V. Paxson, W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.
- [26] W. Feng, D. Kandlur, D. Saha, K. Shin, "A Self-Configuring RED Gateway", Proceedings of INFOCOM 99.
- [27] S. Floyd, K. Fall, "Router mechanisms to support end-to-end congestion control", February 1997. LBL Technical Report.
- [28] Sally Floyd, Kevin Fall, "Promoting the Use of End-to-End Congestion Control in the Internet", IEEE\slash ACM Transactions on Networking, 1999
- [29] P. McKenney, "Stochastic Fairness Queueing". In Proc. IEEE INFOCOM, March 1990.
- [30] R. Jain, "A Comparison of Hashing Schemes for Address Lookup in Computer Networks," IEEE Transactions on Communications, Vol. 40, No. 3, October 1992, pp. 1570-1573.
- [31] M. Shreedhar, G. Varghese, "Efficient fair queuing using deficit round robin", IEEE/ACM Trans. Networking, June 1996

-
- [32] Ion Stoica, Scott Shenker, Hui Zhang, "Core-Stateless Fair Queueing Achieving Approximately Fair Bandwidth Allocations in High Speed Networks", SIGCOMM 1998
- [33] Network Simulator. Available at <http://www.isi.edu/nsnam/ns/>
- [34] Sally Floyd, "RED Discussions of Setting Parameters", Available at <http://www.icir.org/floyd/REDparameter.txt>
- [35] R. Jain, "The Art of Computer Systems Performance Analysis", New York Wiley-Interscience, 1991.
- [36] A. Feldmann, A. Gilbert, P. Huang, W. Willinger, "Dynamics of IP traffic A study of the role of variability and the impact of control", Proceeding of the SIGCOMM, August 1999.
- [37] B. Mah, "An Empirical Model of HTTP Network Traffic", Proceeding of IEEE INFOCOMM, 1997.