석 사 학 위 논 문

IP 망에서의 전송율 측정에 기반한 동적 버퍼 관리

조 정 우 (曺 政 佑)

전자전산학과(전기 및 전자공학전공)

한국과학기술원

2002

IP 망에서의 전송율 측정에 기반한 동적 버퍼 관리

Dynamic Buffer Management Based on Rate Estimation in IP Networks

Dynamic Buffer Management Based on Rate Estimation in IP Networks

Advisor : Professor Dong-ho Cho

by

Jeong-woo Cho

Department of Electrical Engineering & Computer Science
- Division of Electrical Engineering
Korea Advanced Institute of Science and Technology

A thesis submitted to the faculty of the Korea Advanced Institute of Science and Technology in partial fulfillment of the requirements for the degree of Master of Engineering in the Department of Electrical Engineering & Computer Science - Division of Electrical Engineering.

Daejeon, Korea
2001. 12. 27
Approved by

Professor Dong-ho Cho
Major Advisor

IP 망에서의 전송율 측정에 기반한 동적 버퍼 관리

조 정 우

위 논문은 한국과학기술원 석사 학위논문으로 학위논문 심사 위원회에서 심사 통과하였음.

2001년 12월 27일

심사위원장 조 동 호 (인)

심사위원 성 단 근 (인)

심사위원 이 황 수 (인)

사랑하는 가족들에게 드립니다.

MEE 20003530 조 정 우. Jeong-woo Cho. Dynamic Buffer Management Based on Rate Estimation in IP Networks. IP 망에서의 전송율 측정에 기반한 동적 버퍼 관리. Department of Electrical Engineering & Computer Science - Division of Electrical Engineering. 2002. 51p. Advisor: Prof. Dong-ho Cho. Text in English.

ABSTRACT

While traffic volume of real-time applications is rapidly increasing, current routers do not guarantee the minimum QoS values of fairness and drop packets in random fashion. If routers provide a minimum QoS, less delays, more fairness, and smoother sending rates, TCP-Friendly Rate Control can be adopted for real-time applications. We propose a dynamic buffer management scheme that meets the requirements described above, and can be applied to TCP flow and to data flow for transfer of real-time applications. The proposed scheme consists of a virtual threshold function, an accurate and stable per-flow rate estimation, a per-flow exponential drop probability, and a dropping strategy that guarantees fairness when there are many flows. Moreover, we introduce a practical definition of active flows to reduce the overhead coming from maintaining per-flow states. Introduction of a virtual threshold function that divides router operation into three modes helps routers support more flows than RED, FRED and DRR with the same buffer size. Moreover, we introduce Fair Drop that guarantees fairness even when there are so many flows that each flow can buffer less than one packet in average in a router buffer.

Contents

List of Figures

1. Introduction

1.1 Background

TCP is the most widely used transport protocol in the Internet and is appropriate for FTP and Telnet, which both require reliability. However, because it uses an Additive Increase Multiplicative Decrease (AIMD) algorithm and induces coarse timeouts, it can neither ensure smoothly-changing sending rate nor can be used for real-time applications [1]. Because most current routers use Drop Tail as a buffer management scheme, which does not guarantee fairness and delay bound, there has been no motivation for real-time applications to use end-to-end congestion control mechanisms. For these reasons, real-time applications use robuster congestion control schemes than TCP congestion control [2]. Even though Drop Tail is a simple buffer management scheme, it tends to penalize bursty traffic, such as TCP traffic, does not guarantee fairness, and adds unnecessary delays because it doesn't drop any packets before the buffer space is fully exhausted.

These problems can be partially solved by using a number of techniques. If a router can maintain a separate queue for each flow, per-flow queueing schemes, such as FQ, SFQ, and DRR can be used [3, 4, 5]. Although these schemes solve many problems, they require a router to maintain a separate queue for each flow. Moreover, per-flow queueing and per-flow scheduling are very complex to be implemented. Furthermore, FQ requires huge buffer size to support many flows. For example, to support one thousand flows, FQ requires a router to keep several Mbytes buffer size assuming that each IP packet size is about one kbytes. Although SFQ reduces overhead caused by mapping from source-destination address pair to the corresponding queue, it requires even larger router buffer size than FQ to guarantee a comparable fairness compared to FQ. Moreover, in the present situation, most of routers use a single first-in first-out (FIFO) buffer shared by all flows.

Adopting a single FIFO buffer, CSFQ (Core-Stateless Fair Queueing) [6] uses per-flow state only in edge routers. Entering the network, packets are marked with an estimate of their sending rate. A core router compares the estimate of each rate with the fair share of that flow and preferentially drops packets if the flow arrives at a higher rate than its fair share. Although CSFQ is much fairer, it requires an extra field in the IP header of every packet and CSFQ must be installed in contiguous fashion on routers.

RED (Random Early Detection) [7] and FRED (Flow Random Early Drop) [8] are the foundation of buffer management schemes because they are practicable and are designed in the consideration of burstiness of TCP flows. RED prevents full exhaustion of buffers and drops packets before congestion becomes severe. However, it does not prevent unresponsive flows from monopolizing buffer space, and TCP-friendly flows attain only a fraction of their fair share [9]. Also, it can not control queue size effectively and can not prevent buffer overflow when there are many flows [10]. To address the problem of unresponsive flows, in [9], authors stressed on the need for end-to-end congestion control. Furthermore, they insisted that there should be some mechanism on the network to identify and regulate unresponsive flows. Techniques to identify and punish unresponsive flows have been identified in [11, 12]. While these proposals are simple and feasible schemes that solve the problem of unresponsive flows, they can punish unlucky TCP-friendly flows with non-zero probability. Therefore, we do not think that these schemes can be adopted in the present situation. FRED uses a per-flow state to solve the problem of unresponsive flows. Although FRED can not prevent buffer overflow for many flows, it is much fairer than RED and effectively regulates unresponsive flows.

Although RED and its variants can be satisfactory for applications that only require reliability, real-time applications require a router to provide more functions. Moreover, to motivate real-time applications to use TFRC (TCP-Friendly Rate Control) [13, 14, 15, 16] a minimum QoS (Quality of Service) should be guaranteed. First of all, a router should be able to eliminate unnecessary queueing delays because multimedia applications do not want

to experience large queueing delays. Second, a buffer management scheme should be able to regulate unresponsive CBR and UDP flows not to take unfairly large share. Third, a router should support more flows fairly with limited buffer size because IP packet size is relatively large and such a large packet size requires large buffer and results in longer queueing delays. To solve these problems, we propose a new buffer management scheme that ensures better fairness between TCP-friendly flows and unresponsive flows, less delays, and smoother sending rates.

1.2 Chapter Organization

The organization of this paper is as follows: First, we review some buffer management schemes previously proposed in Chapter 2. In Chapter 3, we discuss general requirements of buffer management schemes in packet-switched networks and the detailed algorithm we propose is explained with a discussion of operation mechanics. In Chapter 4, we show simulation results obtained using our proposed scheme, RED, FRED, and DRR, and analyze the results. An analysis of various topics relating to our scheme is also presented in Chapter 4. Finally, we present a conclusion in Chapter 5.

2. Related Works on Buffer Management in IP Networks

To understand how congestion control in IP networks operates, we should understand how TCP congestion control operates because TCP congestion control has many impacts on buffer management scheme in IP routers. Therefore, we describe briefly about TCP congestion control.

TCP evolves its congestion window *cwnd* according to network condition. In Figure 2.1, we can look at how the congestion window evolves throughout the lifetime of a TCP connection. We assume that the TCP receive buffer is so large that the receiver window does not limit evolution of the congestion window.

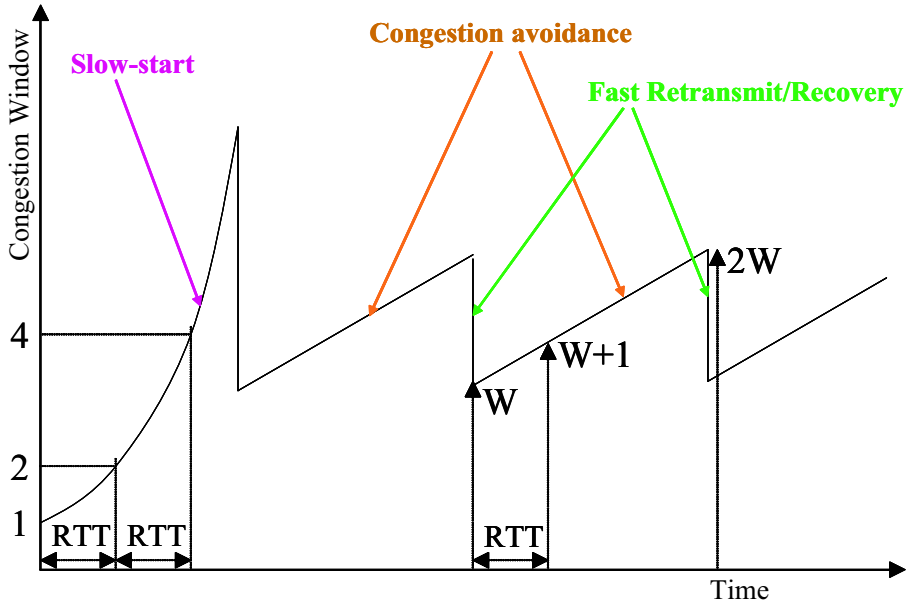


Figure 2.1: Evolution of TCP's congestion window

At connection establishment, the congestion window is initialized to 1 MSS (Maximum Segment Size) and **slow start** mode begins its operation. After one RTT (Round-trip Time), the window is increased to 2 MSS. In this way, the window is multiplied by 2 for every RTT. Therefore, the congestion window increases exponentially fast.

The slow-start ends when the window size exceeds the value of *ssthresh*. When the congestion window is larger than *ssthresh*, the congestion window increases linearly by 1 MSS. This operation has the effect of increasing the congestion window by 1 MSS in each RTT. This phase is called **congestion avoidance** mode. But the window size will not increase forever because TCP sending rate will be in such condition that one of the links in the network will be saturated, at which point packet loss(or drop) will occur. This invokes a timeout. When a timeout occurs, the value of *ssthresh* is set to half the value of the current congestion window, and the congestion window is reset to 1 MSS. Because TCP increases its window size by one for each RTT when its network path is congested, and decreases its window size by a factor of 2 for each RTT when the path is congested, TCP is referred to as an **additive-increase, multiplicative-decrease (AIMD)** algorithm.

Because frequent timeouts can induce low utilization of network capacity and can cause unfairness for TCP flows, TCP Reno and TCP Newreno have employed **fast retransmit** mechanism and **fast recovery** mechanism. The main effect of these two mechanism is in that they can avoid slow start mode and only congestion avoidance mode will be repeated. To induce fast retransmit and fast recovery, the average size of the congestion window should be at least four as shown in [17]. This feature of TCP inevitably requires routers in the network to have large buffers in the output links.

There are many problems that is caused by TCP's congestion control algorithm. First of all, TCP detects congestion only by packet drop. Therefore, routers should drop packets to control each TCP flow's queue size and congestion. This means that there is not explicit feedback between a TCP flow and routers on the flow's path but only packet drop. Furthermore, error recovery algorithm and congestion control algorithm in TCP can not be sep-

arated because TCP uses its congestion control algorithm for both of error recovery algorithm and congestion control algorithm.

Many network applications run over TCP rather than UDP because they want to utilize TCP's reliable transport service. But many multimedia applications do not run over TCP because they do not want their transmission rate throttled, even though the network is severely congested. These flows are called unresponsive flows. Because these applications' congestion control algorithms do not cooperate with TCP congestion control and they do not reduce their transmission rates appropriately, there are severe unfairness problems in current IP networks.

2.1 RED (Random Early Detection)

Traditional routers used in past years had no specific buffer management scheme and most of them used Drop Tail as their buffer management schemes. Routers with Drop Tail drop packets when the buffer space is fully exhausted and they do not drop any packets when there are space for buffering. Therefore, Drop Tail does not actively control queue sizes in buffer space and queueing delays are not controlled. RED (Random Early Detection) [7] was proposed to actively control queueing delays and to avoid global synchronization by randomizing packet drops. When Drop Tail is used for TCP flows, the instantaneous queue size overflows and underflows such that buffer space is always fully exhausted or empty. This causes underutilization and unfairnes problem for TCP flows. Because RED drops packets even if buffer space is not fully exhausted and drops packets according to the grade of congestion, they can avoid underutilization and buffer overflows to some extent. Furthermore, they can control average queue sizes and queueing delays by dropping packets according to estimated average queue sizes. Because RED uses a single FIFO buffer and its algorithm is very simple, RED can be implemented easily and there have been many researches improving or modifying RED algorithm.

RED estimates average queue size by Exponential Weighted Moving Av-

erage (EWMA) as follows:

$$avg_q = (1 - w_q) \times avg_q + w_q q. \tag{2.1}$$

where q is the instantaneous queue size, avg_q is the average queue size estimated by EWMA and w_q determines the time constant of the low-pass filter.

RED drops packets according to the estimated average queue size with the drop probabilities that is depicted in Figure 2.2. As shown in this figure, drop probability for packets increases linearly as the average queue size increases. Drop Probability p_b is calculated as shown in Figure 2.2 and it is recalculated to drop packets uniformly and to prevent consecutive drops.

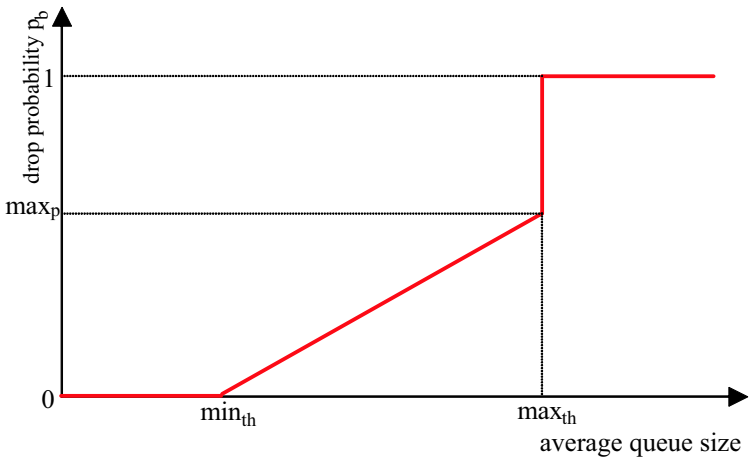


Figure 2.2: Drop probability of RED algorithm

RED suggests that min_{th} should be set to one fourth of physical buffer size to avoid underutilization and that max_{th} should be set to half the value of physical buffer size to buffer bursty traffic. When the average queue size is smaller than min_{th}, no packet is dropped. When the average queue size is larger than max_{th}, every arriving packet is dropped. When the average queue size is between min_{th} and max_{th}, each arriving packet is dropped according to the average queue size.

max_p stands for the maximum drop probability in case that the average queue size does not exceed max_{th}. A higher value of max_p can reduce queueing delays. But, doing so induces underflows. With a higher value of max_p, RED can achieve a better fairness value in severe congestion condition. As a rule of thumb, max_p is set to 0.1 or 0.2. By setting these parameters properly, RED can maintain the average queue size between max_{th} and min_{th} when the number of flows is not so large and their aggregate input load is not so large.

2.2 FRED (Flow Random Early Drop)

Although RED can actively control queueing delays and can guarantee fairness to some extent, RED has bias against TCP and TCP-friendly flows. Because multimedia applications do not use congestion control mechanism and do not reduce their transmission rates even if network is severely congested, they take a larger bandwidth share than TCP flows. To solve this problem, FRED (Flow Random Early Drop) [8]was proposed.

To protect fragile flows like TCP flows from unresponsive flows like CBR and UDP flows, FRED maintains per-flow states for all flows that have at least one buffered packet in the router buffer.

Calculation of the drop probability p_b and avg_q in FRED is the same as RED. But, FRED do not drop packets with the probability p_b and there are additionally checked conditions in case of FRED. The main difference between RED and FRED is that FRED compares the average queue size $avgcq$ with per-flow queue size $qlen_i$ and drops packets conditionally. This feature can be described in Figure 2.3. It should be noted that FRED uses a single FIFO buffer and per-flow queues in Figure 2.3 are buffered in a single FIFO buffer in fact.

Although FRED uses a single FIFO buffer, it maintains per-flow states. Therefore, FRED can calculate $avgcq$ that is the average per-flow queue size. When the average queue size is smaller than min_{th}, no packet is dropped. When the average queue size is larger than max_{th}, every arriving packet is

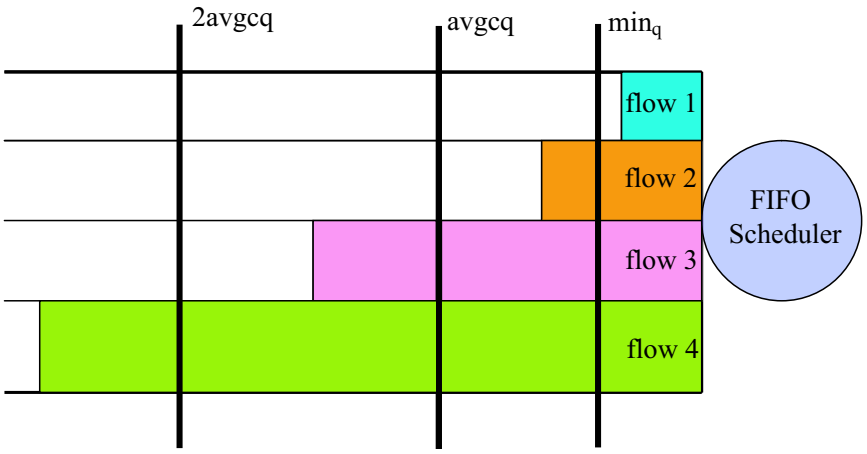


Figure 2.3: Operation of FRED algorithm

dropped. This operation is the same as RED. When the average queue size is between min_{th} and max_{th}, FRED compares the per-flow queue size $qlen_i$ with $avgcq$. If $qlen_i$ exceeds $2avgcq$, FRED drops the arriving packet with probability 1. If $qlen_i$ is smaller than $avgcq$, FRED do not drop the arriving packet. FRED also records the number of over-runs, that is, the times when $qlen_i$ is larger than $avgcq$. If a flow induces over-runs more than 1 time, FRED drops the arriving packet with probability 1.

In Figure 2.3, flow 1 and 2 can be classified as well-behaving flows and flow 3 and 4 can be classified as unresponsive flows. Furthermore, arriving packets of flow 1 and 2 will not be dropped and arriving packets of flow 3 will experience random drop like RED because the per-flow queue size of flow 3 exceeds $avgcq$ and does not exceed $2avgcq$. Arriving packets of flow 4 will be dropped unconditionally because per-flow the queue size of flow 4 exceeds $2avgcq$. Details of FRED algorithm can be found in [8].

In this way, FRED succeeds in protecting TCP flows from CBR flows. But, there are many problems that can not be solved by FRED. One of the problems of FRED is that FRED is hard to be analyzed mathematically and its performance is not verified over wide range of network configurations.

Second, FRED can not actively control the average queue size because it do not drop the arriving packets sent by well-behaving TCP flows before the average queue size grows large. This can result in longer queueing delays. Third, since FRED only keeps state for flows which have packets queued in the router buffer, it requires a large amount of buffers to work well. Without sufficient buffer space, it becomes hard for FRED to detect unresponsive flows since they may not have enough packets continually buffered to trigger the detection mechanism. In addition, unresponsive flows are immediately reclassified as being responsive as soon as they clear their packets from the router buffer.

2.3 DRR (Deficit Round Robin)

In ATM networks, there have been many sophisticated scheduling disciplines such as GPS (Generalized Processor Sharing) , WFQ (Weighted Fair Queueing) [3], WF^2Q (Worst-case Fair Weighted Fair Queueing), DRR (Deficit Round Robin) [5] and SCFQ (Self-clocked Fair Queueing). Among these disciplines, DRR has attracted researchers in IP networks for its ease of implementation. Although DRR is much simpler than other scheduling disciplines in ATM networks, DRR is still more complicated scheme than any other schemes such as RED and FRED in IP networks. As a packetized version of WFQ, DRR scheduler associates each flow with a **deficit counter** initialized to 0. The scheduler visits each flow in turn and tries to serve one **quantum** worth of bits from each visited flow. The packet at the head of the queue is served if it is no larger than the quantum size. If it is larger, the quantum is added to the flow's deficit counter. If the scheduler visits a flow when the sum of the flow's deficit counter and the quantum is larger than or equal to the size of the packet at the head of the queue, then the packet at the head of the queue is served, and the deficit counter is reduced by the packet size. An example operation of DRR is shown in Figure 2.4.

It should be reminded that DRR is only a scheduling discipline and is not a dropping strategy. Therefore, its performance can be degraded when

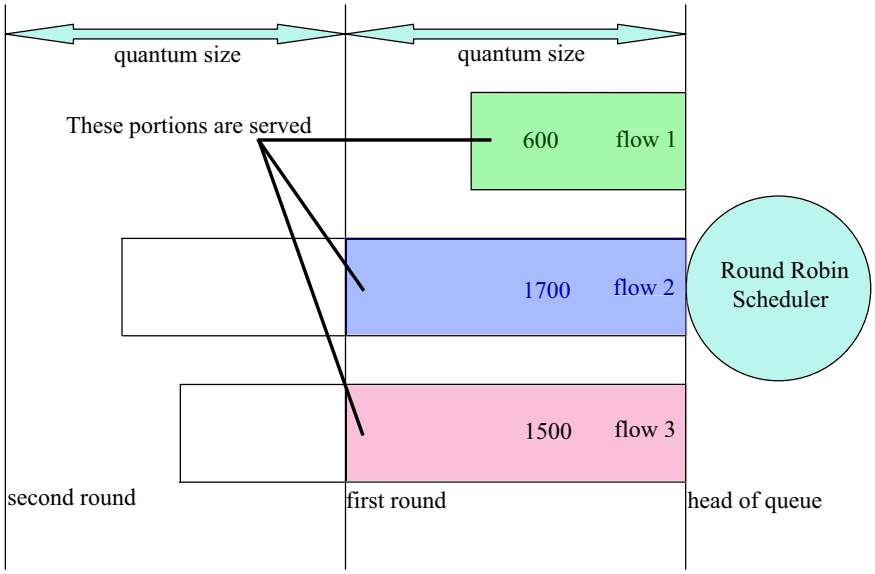


Figure 2.4: Operation of DRR algorithm

it is not used with a proper dropping strategy. Recently, it is revealed that DRR should be used with the proper dropping strategy called LQD (Longest Queue Drop) [18] to be used in IP networks. Furthermore, FIFO scheduling with LQD is shown to be superior to DRR with RED in many aspects.

The motivation for LQD is that if flows are given equal weights, then flows which use the link more tend to have longer queues. Hence, biasing the packet drops such that flows with longer queues have higher drop rates should make the bandwidths sharing more fair. For proper operation of LQD, LQD should maintain the flow number that has the longest queue in the current router buffer. This requires $O(log\ N)$ (where N is the number of flows) time complexity. For each arriving packet, LQD determines whether the buffer space is fully exhausted or not. If the buffer space is not fully exhausted, LQD does not drop any packets. If the buffer space is fully exhausted, LQD drops the packet at the head of the longest queue and buffers the arriving packet. This feature of LQD allows TCP flows to invoke fast retransmit and fast recovery. Hence, LQD makes bandwidth sharing of TCP flows even more

fairer.

Although DRR shows extremely good performance in many cases, there are two main problems that DRR can not solve. One of the most important problems is that DRR requires $O(log\ N)$ time complexity in both mapping the arriving packet to the proper buffer number and finding the flow number of the longest per-flow queue. It also requires per-flow buffering that is hard to be implemented in current IP networks. Second problem is that DRR guarantees fairness only for backlogged flows. When the number of flows exceeds the maximum number of per-flow buffer in routers, DRR can not guarantee fairness.

3. Proposed BMRE Algorithm

RED is a simple and powerful buffer management scheme that drops packets from each flow in proportion to the amount of bandwidth the flows used on the output link [8], assuming that all flows exhibit the same behavior as TCP flows do in view of packet drop events. However, RED cannot prevent buffer overflow for many flows, cannot regulate unresponsive flows, and is unfair even among TCP flows because it drops packets randomly [8, 9, 10, 11]. We suggest the following functions that an intelligent buffer management scheme should support:

1. Low queueing delays
2. Control of the queue size to prevent overflow and underflow
3. Regulation of unresponsive flows and fairness
4. Smooth sending rates for each flow

In this paper, we define "flow" as a source-destination IP address pair to distinguish each flow transferring multicast traffic data and to guarantee fairness for those flows. For example, address pairs (A, B) and (C, B) are treated as different flows. Although definition of "flow" can be extended to each TCP port or UDP port, currently, header processing of layer 4 in routers is not common.

3.1 A Scalable and Fair Buffer Management Scheme

In ideal situations, routers can provide fairness even with a small buffer. But, TCP, which is dominant transport protocol, requires a large buffer because it uses window-based congestion control that causes frequent coarse timeouts when there is insufficient buffer space. This results in short-term unfairness. Although TCP flows require that at least 4 packets per flow

should be buffered in routers to prevent coarse retransmit timeouts [17], most routers provide very small buffers because large buffers without an active buffer management scheme results in unacceptably long delays and long response times.

To guarantee fairness with TCP flows, a buffer management scheme should allow each flow to buffer at least 4 packets when congestion is not severe. However, just allowing each flow to buffer at least 4 packets can be unfair when TCP flows and unresponsive flows(ex. CBR flows) coexist. To alleviate this unfairness, a buffer management scheme also should drop packets according to each flow's estimated throughput.

A router do not provide a large buffer because a large buffer inevitably results in longer queueing delays. Therefore, a buffer management should gracefully adjust per-flow queue sizes according to the number of active flows. When congestion is severe, for example, in case that there are ten thousand flows and the router buffer size is one thousand kbytes, a flow can buffer only 100 bytes in average. Assuming that an IP packet size is 500 bytes, a flow can buffer only 0.2 packets in average. Guaranteeing fairness in such a condition is not easy due to the following reasons: (1) In such a situation, estimating per-flow rates is not an easy task. Because TCP's retransmit timeout value is doubled for every consecutive retransmit timeout, estimating per-flow rates and guaranteeing fairness in such a situation are difficult. (2) Maintaining several millions of per-flow states in a router is also not an easy task. If per-flow states are implemented in conventional memory such as RAM, mapping from source-destination address pair to the corresponding state requires $O(log\ N)$ (where N is the number of flows) time complexity. If a new and practical definition of flows could reduce this complexity, it would be feasible for routers to maintain such a reduced number of per-flow states.

To minimize unnecessary queueing delays, to guarantee fairness, and to allow a flow to buffer at least 4 packets, we propose a virtual threshold function, shown in Figure 3.1. In this figure, we divide router operation into three modes. Each flow can buffer up to $vmax_q/N_{flow}$ bytes. Because each TCP flow does not occupy more than $vmax_q/N_{flow}$ bytes all the time,

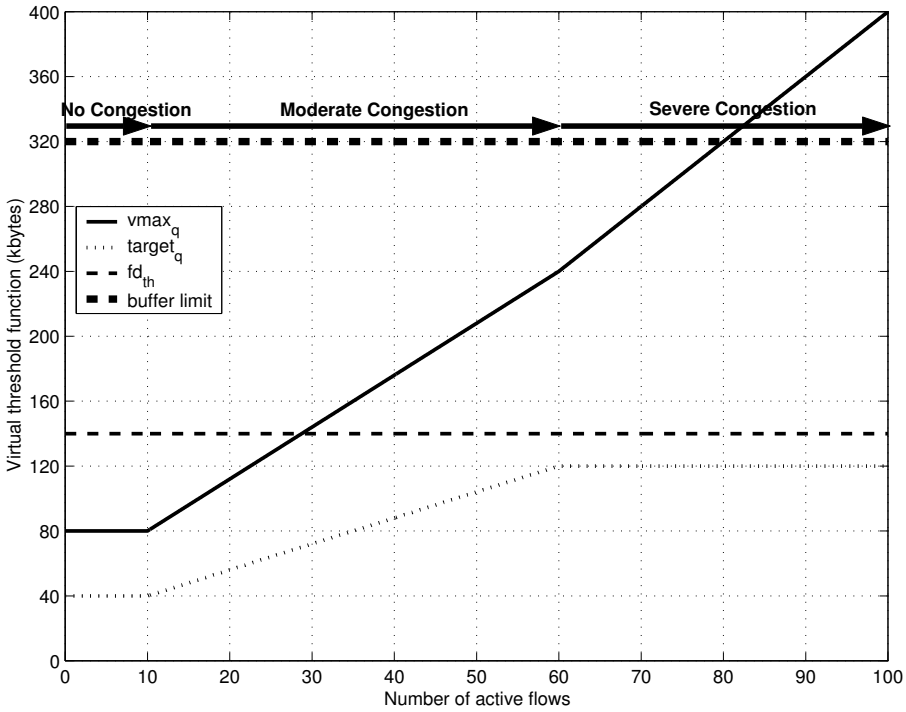


Figure 3.1: Virtual threshold function vs. number of active flows

exploiting the burstiness of TCP, we can maintain an average queue size to a target value, which is shown as $target_q$. **In no congestion mode**, there is sufficient buffer space to allow each flow to buffer at least 8000 bytes. In this mode, a router can provide highly satisfactory QoS. **In moderate congestion mode**, there is insufficient buffer space and the queueing delay increases. In this mode, we allow each flow to buffer a smaller number of packets as the number of flows increases. **In severe congestion mode**, each flow can buffer only a minimum number of packets, that is to say, 4000 kbytes, and we can not provide low delays. We also introduces a dropping strategy named "Fair Drop" that guarantees fairness when there are many flows. Fair Drop operates when queue size is larger than fd_{th}. A buffer management scheme should limit router's queue size to a certain value to prevent buffer overflows. Fair Drop is designed to prevent buffer overflows and to limit maximum queueing delays still maintaining satisfactory fairness values.

As demands on delay and per-flow buffer size can vary, the virtual threshold function can also vary according to these demands.

3.2 Why Should a Router Drop Packets Periodically?

Achieving smooth sending rates requires periodic dropping of packets. However, RED drops packets randomly as shown in this section.

TCP packet losses are detected based on the following two ways: (1) The TCP sender can detect them either when it receives *triple-duplicate* acknowledgements, (four ACK's with the same sequence number), or (2) when *retransmit timeouts* occur [19]. We define the *congestion cycle* CC_i as the ith period between two loss indications and define α_i as the number of packets including the first packet loss in CC_i. If RED is in steady state, which means no recent change in the number of flows, packets for flow i are dropped with nearly constant drop probability p. Therefore, α_i is distributed geometrically as follows:

$$P[\alpha_i = k] = (1-p)^{k-1}p, \quad k = 1, 2, \cdots. \tag{3.1}$$

As can be seen from this equation, each flow experiences geometrically distributed inter-drop times. The mean and standard deviation of α_i are as follows:

$$E[\alpha_i] = \sum_{k=1}^{\infty}(1-p)^{k-1}pk = \frac{1}{p}, \tag{3.2}$$

$$S[\alpha_i] = \frac{1}{p}\sqrt{1-p}. \tag{3.3}$$

We can determine that $E[\alpha_i] = 10$ and $S[\alpha_i] = 9.5$ with $p = 0.1$, indicating that some flows buffer more than a sufficient number of packets and others buffer fewer than the necessary number of packets. This feature of RED causes unfairness, inefficient buffer usage, and rough sending rates. To avoid these problems, routers should drop packets periodically.

3.3 BMRE Algorithm

We propose BMRE (Buffer Management based on Rate Estimation) scheme which solves the problems discussed in Section 3.1 and 3.2. BMRE consists of virtual threshold function that eliminates unnecessary queueing delay and prevents buffer overflows and underflows, an accurate per-flow rate estimation that measures each flow's rate, a per-flow exponential drop probability that keeps TCP flows from unresponsive flows, and Fair Drop that guarantees fairness even when the number of flows is very large.

BMRE's basic algorithm is depicted in Figure 3.2. To understand the detailed operation of BMRE algorithm, it is necessary to refer to the detailed pseudocode of BMRE algorithm in Appendix.

(1) For each packet's arrival, global queue size q is compared with $vmax_q$ to prevent unnecessary fluctuation of global queue size q and is compared with max_q to prevent buffer overflows. **(2)** If there is no flow state for this packet, BMRE assigns a flow state to this flow. Because a buffer management scheme can support only a finite number of flow states, when all flow states are occupied, BMRE assigns a used flow state whose per-flow queue size is 0 to this new flow. Because global queue size q is controlled not to exceed half of buffer size BS by BMRE, if the maximum number of per-flow states max_{flow}

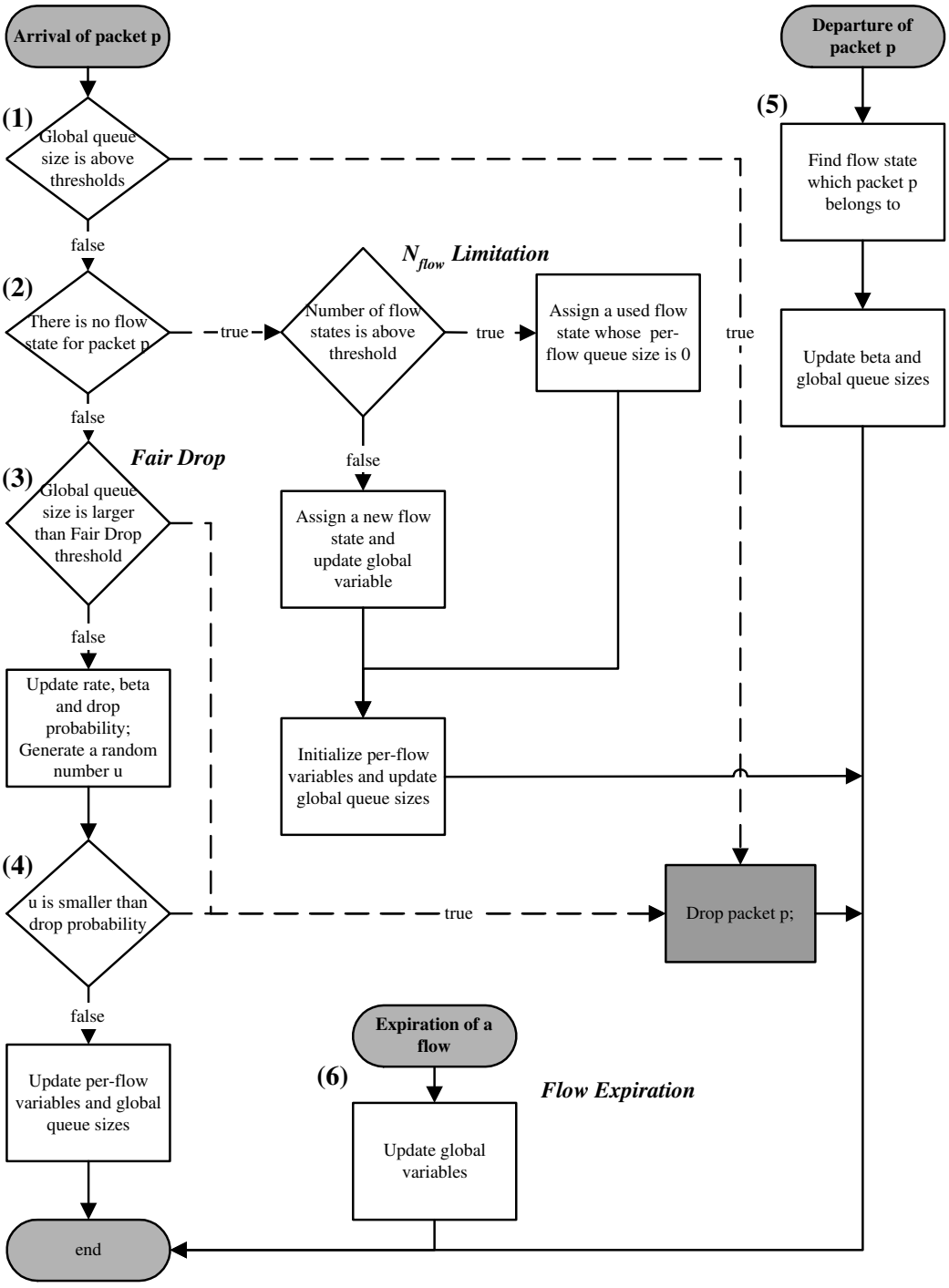


Figure 3.2: BMRE algorithm

is set to a proper value, at least one per-flow state would have 0 per-flow queue size. Therefore, a packet whose per-flow state is not registered in per-flow states is never dropped. **(3)** Because a buffer management scheme should regulate global queue size q to prevent buffer overflows, when global queue size q is larger than Fair Drop threshold fd_{th}, an arriving packet is dropped. **(4)** BMRE drops packets according to the estimated per-flow rate. **(5)** For each packet's departure, BMRE finds the flow number of this departing packet and updates beta $\beta[i]$ and global queue sizes. **(6)** If flow i's per-flow queue size is 0 for more than timeout value $Timeout\_Value$, flow i expires and the flow state for the flow is deleted.

BMRE determines a maximum per-flow buffer size depending on the number of currently active flows and drops packets based on the rate estimation of each flow [6]. As an estimate of per-flow share, either a per-flow average queue size estimation in [7] or a per-flow rate estimation in [6] can be used. In fact, using per-flow average queue size requires replacing "$\mathbf{exp}(-dt/K)$" with a constant "λ" and replacing rate estimates with average queue estimates in code line `31` in Appendix as follows. (In addition to this replacement, a portion of code should be modified.)

$$rate[i] \leftarrow (1 - e^{-dt/K}) \times \frac{p.size}{dt} + e^{-dt/K} \times rate[i], \tag{3.4}$$

$$avg_q[i] \leftarrow (1 - \lambda) \times q[i] + \lambda \times avg_q[i]. \tag{3.5}$$

The per-flow buffer occupancy of flow i is proportional to the per-flow output rate of flow i with the FIFO discipline [8]. Therefore, we can guess that these two approaches achieve the same performance. However, the usage of per-flow average queue size as an estimate of the per-flow share is not as precise and efficient as that of per-flow rate estimation. When per-flow average queue size is used as an estimate of per-flow share, filtering of unnecessary noise and quick responsiveness to rapid rate fluctuations can not be obtained simultaneously. Let assume that end of congestion cycle CC_i is caused only by triple-duplicate ACKs, there are only periodic packet losses, and the round trip time is fixed to RTT. W_i is the maximum window size in Congestion Cycle CC_i. With these assumptions, the inter-packet buffering

time of TCP varies from RTT/W_i to $(2 \times RTT)/W_i$, so the per-flow share can not be calculated accurately without dependency on dt. If there are substantial packet losses caused by timeouts, this discrimination becomes more significant. Therefore, we have chosen to use rate estimation as a method for estimating per-flow share. Rate estimation in code line `31` in Appendix is robust to various packet length distributions and is proven to asymptotically converge to the real rate [6].

3.4 Per-flow Exponential Drop Probability

As shown in code line `24` in Appendix, BMRE drops packets for flow i with following drop probability:

$$p[i] = \left(\frac{q[i]}{max_{th}}\right)^{\beta[i]}. \tag{3.6}$$

Based on per-flow rate estimation and comparison of current average queue size with $target_q$, BMRE either increases or decreases $\beta[i]$. Flow i experiences a high drop probability with a small $\beta[i]$ and experiences a low drop probability with a large $\beta[i]$. Upon decrease, $\beta[i]$ is divided by the $rate_{ratio}$. Upon increase, $\beta[i]$ is multiplied by the constant value of α.

By introducing $\beta[i]$ as an exponent of drop probability for each flow, drop probability of each flow can be adjusted efficiently as shown in Figure 3.3. **First**, $\beta[i]$ is large when i is a TCP or TCP-friendly flow. In this case, using $\beta[i]$ as an exponent of drop probability makes each TCP and TCP-friendly flow experience periodic packet drops. Because TCP is very sensitive to small drop probability such as 0.1, using $\beta[i]$ as an exponent can prevent unnecessary packet drops for flows using smaller share than fair share. For example, if the sending rate of flow i does not exceed fair share (when $q[i] \leq max_{th}/2$), packets of flow i are dropped with a negligible probability, i.e., $0.5^{10.0} = 0.001$. Therefore, BMRE's dropping strategy does not drop flow i's packets when the number of of buffered packets for flow i is less than the average number of buffered packets for other flows. With this dropping method, packets with low rate flows are rarely dropped. **Secondly**, $\beta[i]$ is

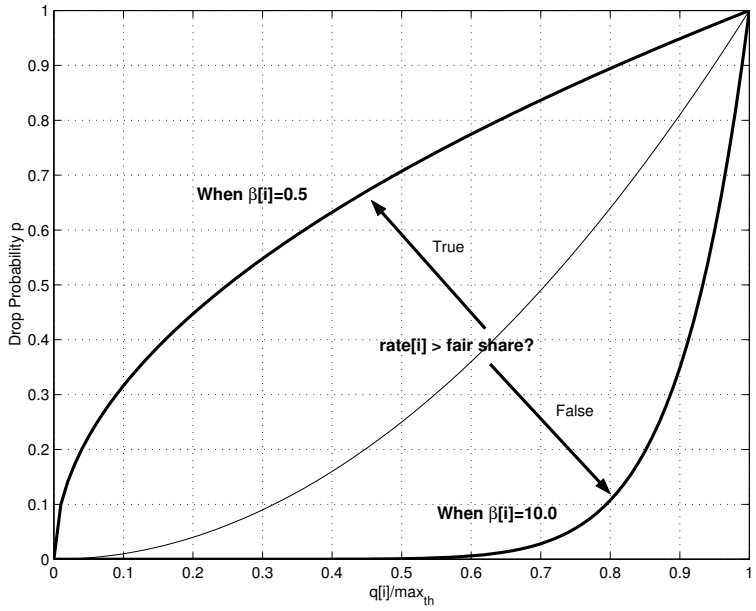


Figure 3.3: Exponential drop probability

small when flow i is unresponsive and is using more than its fair share. In this case, using $\beta[i]$ as an exponent of drop probability allows high drop probability such as 0.9.

With per-flow exponential adjustment of drop probability, we can achieve a high degree of fairness and smooth sending rates because packets of TCP-friendly flows are dropped nearly periodically. Furthermore, the queue size of each flow is well regulated and each flow is not allowed to buffer more than the necessary number of packets.

3.5 Choosing K

The choice of decay factor K involves several tradeoffs. First, while a smaller K value increases the system responsiveness to rapid rate fluctuations, a larger K value better filters noise and avoids potential system instability. Second, K should be large enough to smooth the estimated sending rates of TCP flows because these rates are estimated to be high when TCP

flows have large window sizes just before packet drop events. To control these effects, K can be decided as follows:

$$K \leftarrow C \times \frac{Average\ Packet\ Size \times N}{BW}. \tag{3.7}$$

where N can be substituted for N_{flow} and $Average\ Packet\ Size$ is the average IP packet size, and BW is the link speed or service rate of a router. We found that C should be $10 \sim 30$ through numerous simulations and overall performance is very insensitive to various K values. But, (1) using a higher value of K requires a router to maintain unacceptably large number of per-flow states, and (2) guaranteeing fairness when there is insufficient buffer space such that each TCP flow can buffer only up to 0.1 packets in average is meaningless. Moreover, (3) frequent change of K can induce implementation complexity. Considering these three points, as a rule of thumb, we recommend that K should be $1 \sim 3$ times value of the average queueing delay, which can be calculated based on dividing the average queue size by the link speed.

3.6 Practical Definition of Active Flows

Because routers have a limited memory, per-flow states should be deleted properly, but neither too often nor too seldom. With too frequent deletion of the per-flow states, the defects caused by FRED's frequent deletion of per-flow states can appear. FRED's defects due to the frequent deletion allows unresponsive flows to take more than fair share because number of active flows are underestimated and fair share is overestimated. With too infrequent deletion of the per-flow states, the number of active flows are overestimated and fair rate can be underestimated so that all flows could be dropped simultaneously. From code line `31` in Appendix, the rate of flow i is updated according to (3.4). In (3.4), if $e^{-dt/K}$ is set to $e^{-3} = 0.0498$, dt should be $3K$ and (3.4) becomes as follows:

$$rate[i] \leftarrow 0.9502 \times \frac{p.size}{dt} + 0.0498 \times rate[i] \approx \frac{p.size}{dt}. \tag{3.8}$$

Therefore, routers do not have to maintain the per-flow state of flow i if $dt = 3K$ seconds has elapsed since the last buffering operation of flow i. Therefore, $Timeout\_Value$ used for deleting per-flow states is set to $3K$. This can greatly reduce the overhead coming from maintaining per-flow states.

Although there have been many approaches to buffer management schemes with usage of per-flow states, maintaining per-flow states has been considered to be impractical and not scalable. But, we found that we can reduce the overhead of maintaining per-flow states. **Our main motivation for definition of active flows is that a very weak flow does not need to be regarded as an active flow**. For example, assume that a router's link speed is 20 Mbps, a router buffer size is 160 kbytes, an IP packet size is 1 kbytes, K is 150 ms, and $Timeout\_Value$ is 450 ms. $Timeout\_Value$ can be regarded as a time window because a flow that has not been buffered for $Timeout\_Value$ is deleted. This corresponds to maximum queueing delay of 64 ms and average inter-packet time of 0.4 ms. Because a flow that has not been buffered for more than $Timeout\_Value$ is too weak to be regarded as a flow, we can decide that up to $(450 + 64)/0.4 = 1,285$ flows need to be regarded as active flows. For another example, if a router supports an OC-12c link that has a capacity of 622 Mbps and a 4 Mbytes buffer, and an IP packet size is 1 kbytes, this corresponds to maximum queueing delay of 51.4 ms, K of 60 ms, $Timeout\_Value$ of 180 ms, and average inter-packet time of 12.9 μs. Therefore, this router has to maintain only up to $14,000$ per-flow states. Although this number of per-flow states may be considered as a large number of flows, it is well known that current ATM switches support at least $64,000$ VCs [20]. Furthermore, numbers of flows that should be treated as active flows is still overestimated in the above calculations because the probability that every packet belongs to each distinct flow in a time window is low due to TCP's burstiness.

3.7 Fair Drop

Although our new definition of active flows greatly reduced the overhead of maintaining per-flow states, maintaining such a large number of flows results in longer delays. If we allow 1 kbytes per an active flow with a 20 Mbps link to support $1,500$ flows, it corresponds to a buffer size of 1.5 Mbytes and maximum queueing delay of 0.6 seconds that is so long that flows transferring real-time application data would complain. So, such a large buffer is not likely to be used in practice. To become more realistic, a router would try to support such a large number of flows with a smaller buffer so that each flow can buffer only a fraction of a packet such as 0.2 or 0.5 packets in average. As discussed in Section 3.2, guaranteeing fairness for a large number of flows with a small buffer is not trivial. To guarantee fairness with a small buffer, we introduce Fair Drop scheme. **The main motivation for Fair Drop is that recently buffered flows do not need to be buffered again if the buffer size is not sufficiently large and all flows can not be buffered simultaneously**. When the global queue size is above a specified threshold value, Fair Drop scheme drops packets of flow i if the flow has been buffered recently and its flow state is still maintained by BMRE.

4. Simulation Results and Discussion

BMRE, RED, FRED, and DRR are compared based on simulation results. While RED is selected as a fundamental scheme due to its simplicity, FRED and DRR are selected as comparable schemes with BMRE.

- RED (Random Early Detection) - This scheme is significantly more sophisticated than Drop Tail and is designed for routers with a single FIFO buffer. RED drops packets before congestion becomes severe and controls the average queue size between max_{th} and min_{th} values. When the average queue size is less than min_{th}, there is no packet drop. When the average queue size is greater than max_{th}, all packets are dropped. When the average queue size is between two thresholds, the packet drop probability is increased linearly in proportional to the average queue size.

- FRED (Flow Random Early Drop) - This is an extended version of RED for partial solution of the problem of unresponsive flows. FRED maintains per-flow states for all flows that have a non-zero queue size in the router buffer. Using this per-flow state, FRED preferentially drops the packets of flows that have queue sizes larger than the average per-flow queue size. It unconditionally drops the packets of flows that (1) have queue sizes two times greater than the average per-flow queue size or (2) experience many packet drops. It randomly drops the packets of flows that have larger queue size than the average per-flow queue size with the probability proportional to the average queue size. FRED underestimates the number of active flows and overestimates the per-flow average queue size that is calculated by dividing the average queue size by the number of active flows because FRED deletes per-flow state of

flows that have a zero queue size, and TCP flows in timeouts have no packet buffered. Therefore, FRED encourages smooth-rate unresponsive flows, such as UDP-CBR flows. Furthermore, as the fraction of unresponsive flows increases, the average per-flow queue size itself is increased and unresponsive flows are not well regulated.

- DRR (Deficit Round Robin) - This scheme is a variant of WFQ (Weighted Fair Queueing) discipline. DRR allows WFQ to handle variable packet sizes in a fair manner. DRR is the only one that uses per-flow queueing algorithm while RED, FRED and BMRE use a single FIFO buffer. Therefore, DRR guarantees nearly perfect fairness for flows that have at least one packet in the router buffer. LQD (Longest Queue Drop) is used as a packet drop strategy.

4.1 Simulation Configurations

We simulate the configuration shown in Figure 4.1. Unless otherwise specified, the following parameters are used. Each output link has a capacity of 20 Mbps, a latency of 2 ms, and a single FIFO buffer of 320 kbytes. For RED and FRED, min_{th} is set to 53 kbytes and max_{th} is set to 160 kbytes that corresponds to maximum queueing delay of 64 ms. The buffer size of DRR is set to 320 kbytes.

To compare BMRE, RED, FRED and DRR in a fair manner, max_{th} of DRR is set to 160 kbytes. TCP-Newreno is used in all simulations because it is the most widely used TCP variant as shown in [21] for its robustness against consecutive packet drops [22]. The data packet size of TCP flows is set to 1000 bytes and the ACK packet size is set to 40 bytes. All BMRE parameters are set to the values indicated in Appendix. We limited BMRE's maximum number of per-flow states to 320 to see limiting effects. To avoid the buffer space being fully exhausted, max_p is set to 0.2 for RED and FRED. The FRED min_q value is set to 2000 bytes. All four schemes are implemented in ns-2 [23]. RED and FRED operate in byte mode, meaning that packets are

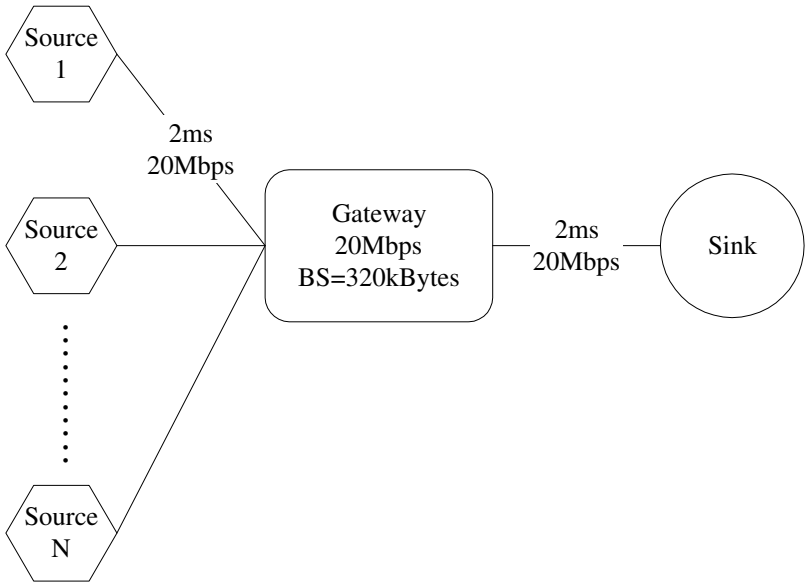


Figure 4.1: Simulation topology

buffered and counted in bytes and dropped with a probability proportional to their size. As an example, 1000 bytes packets is dropped with a probability of 0.2 and 40 bytes packet is dropped with a probability of 0.008 for RED if the average queue size is slightly smaller than max_{th}.

To reduce instantaneous noise and to avoid phase effects, each simulation is run for $T = 100$ seconds and each flow starts at a random time. The term $Goodput_i$ of TCP/CBR/TFRC flow i is defined as the number of bytes received by TCP/CBR/TFRC Sink in unit time. We also define N as the number of all flows that are trying to send data packets in simulations while N_{flow} is an estimated number of flows by BMRE. Consequently, N is always larger than or equal to N_{flow}.

4.2 Performance Behavior of TCP and CBR flows

4.2.1 Queuing Delay and Fairness for TCP flows

We consider only TCP flows and there is no CBR flow in this subsection. As shown in Figure 4.2(a) and 4.2(b), BMRE reduces the unnecessary queueing delays and maintains a much smaller average queue size compared with RED and FRED. In fact, if the average queue size of BMRE is set to $target_q$ of the virtual threshold function, the queueing delays are controlled to the corresponding value. Although the queueing delays increase as the number of flows increases and BMRE shows a slightly longer delay than RED when the number of flows is 50 to 80, BMRE maintains smaller delays compared to RED and FRED in the wide range of numbers of flows. BMRE eliminates unnecessary queueing delays by regulating each flow's queue size based on the knowledge of the number of flows. Queueing delays of RED can be reduced by setting max_p to higher values while queueing delays of FRED can not be noticeably reduced with higher max_p. But, it should be noted that RED does not guarantee fairness as well as QoS, and BMRE outperforms FRED and DRR in all cases.

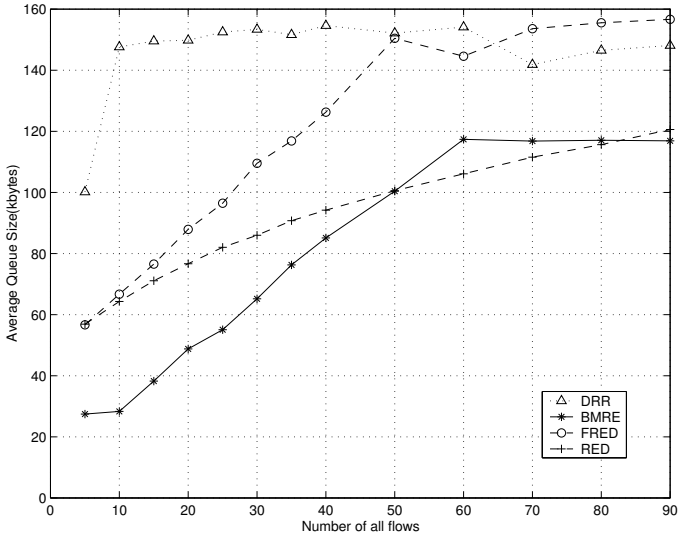
As shown in Figure 4.3(a) and 4.3(b), under the same condition as mentioned above, we measured the standard deviation of the goodput for each flow, which is normalized by the fair share of that flow. The standard deviation S of $(Goodput_i/Fair\ Share)$ is defined as follows.

$$S = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left(\frac{Goodput_i}{BW/N} - 1\right)^2}, \tag{4.1}$$

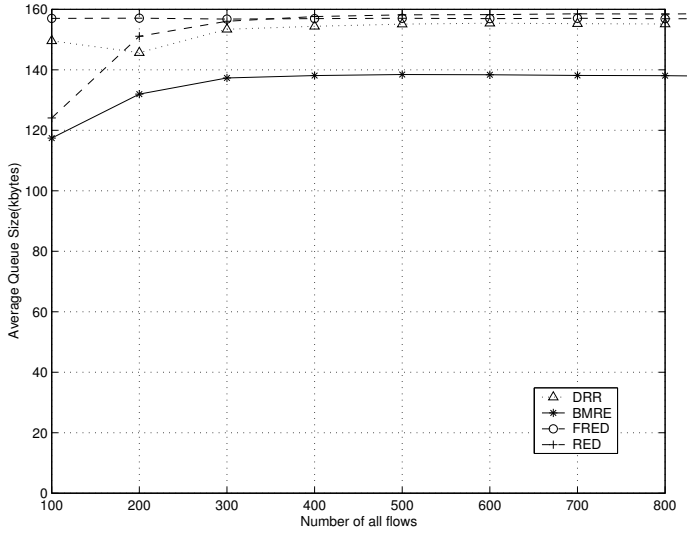
$$Goodput_i = Goodput_i(0, T), \tag{4.2}$$

$$Goodput_i(t, t+\Delta t) = \frac{Total\ bytes\ of\ flow\ i\ received\ in\ [t, t+\Delta t]}{\Delta t}. \tag{4.3}$$

BMRE achieves extreme fairness that can not be compared with RED. When the number of flows is smaller than 100, FRED achieves comparable performance to BMRE. While FRED can support only about 100 flows within standard deviation of 0.1, BMRE support about 350 flows within the same value. BMRE's improved fairness over FRED is largely due to Fair

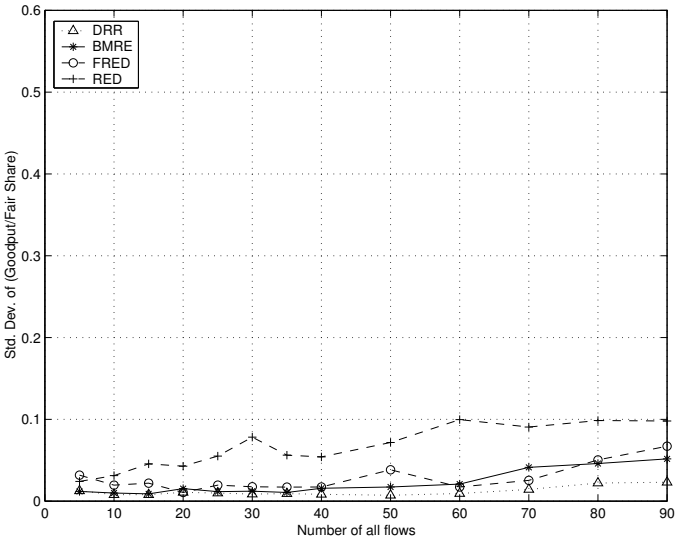


(a) When N is varied from 5 to 90

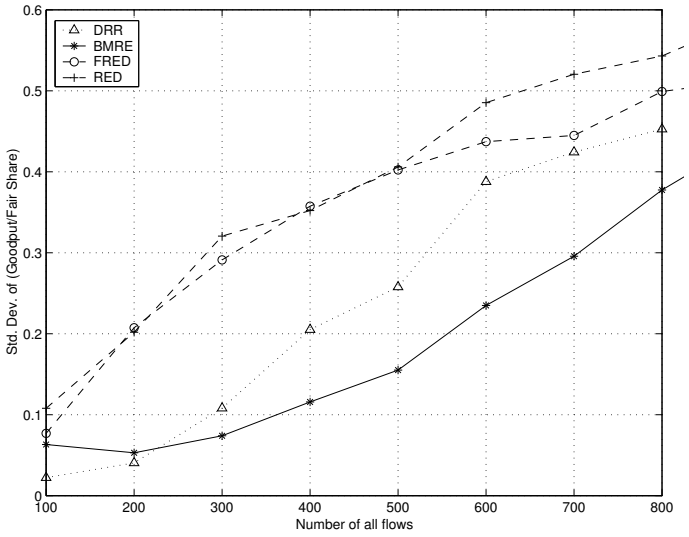


(b) When N is varied from 100 to 800

Figure 4.2: Average queue size vs. number of flows



(a) When N is varied from 5 to 90



(b) When N is varied from 100 to 800

Figure 4.3: $(Goodput_i/Fair\ Share)$ vs. number of flows

Drop. Reasoning from this result, if FRED wants to guarantee fairness values comparable to BMRE, it should maintain three or four times larger buffer compared with BMRE. Therefore, its maximum queueing delay should be three or four times as large as that of BMRE. Although DRR achieves better performance when the number of flows is less than 200, BMRE significantly outperforms DRR when the number of flows is larger than 200 because DRR can ensure fairness only when the number of flows is small and each flow can buffer at least one packet. In contrast to DRR, **BMRE drops packets whose per-flow states are still maintained and makes room for flows that have not been buffered recently.**

Packet loss events are observed with 20 TCP flows. In Figure 4.4, packet loss events of source 1 are shown. BMRE drops packet more frequently than other scheme to minimize queueing delay. RED and FRED drop packets in a random fashion, as expressed in (3.1), while BMRE drops packets nearly periodically. With this periodic packet drop, BMRE can effectively control

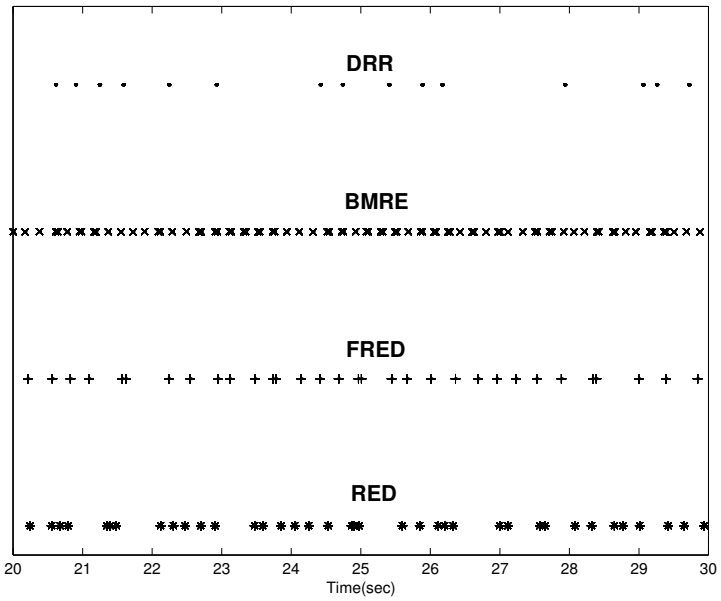


Figure 4.4: Loss events of source 1

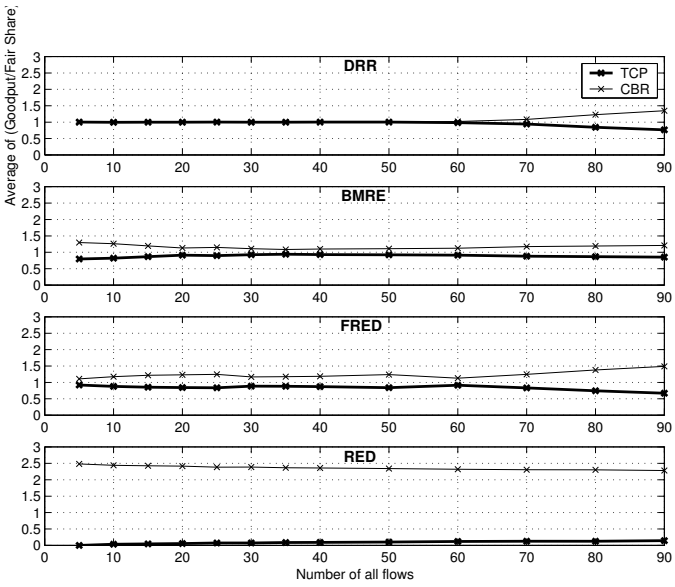
per-flow queue sizes and prevent flow i from buffering more than the necessary number of packets. DRR can not be compared with other schemes because it maintains a per-flow queue for each flow and guarantees perfect fairness for flows that have at least one packets in the router buffer by serving them in round robin discipline.

4.2.2 Fairness for TCP and CBR flows

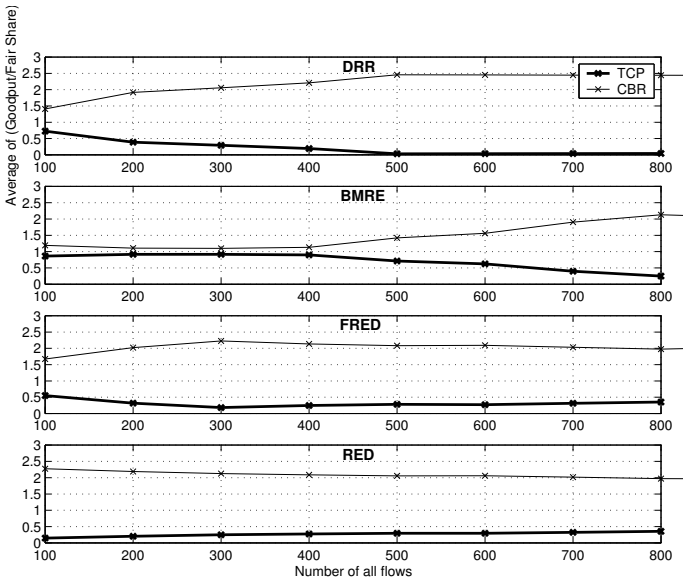
We simulate TCP and CBR (which uses UDP as a transport protocol) flows. In Figure 4.5(a) and 4.5(b), we change the number of flows from 5 to 800. The fraction of CBR flows is set to 40%. Packet size of CBR flows is set to 1000 bytes and inter-packet times are adjusted so that each CBR flows send packets at three times rate of the fair share rate. RED can not protect TCP flows from unresponsive CBR flows at all. BMRE's performance degradation when the number of flows is about 10 is due to TCP's retransmit timeout's. Because TCP's minimum retransmit timeouts are set to 200 ms and round trip times are about 40 ms, TCP flows can not get sufficient share. This problem can be solved by setting $vmax_q$ to a higher value when the number of flows are smaller than 20. BMRE and FRED can protect TCP flows from CBR flows. But, if we decide that TCP flows should receive at least half of their fair share, BMRE can support up to 650 flows fairly due to its Fair Drop while FRED and DRR can support only up to 100 and 200 flows respectively.

Because BMRE knows which flows have been recently buffered and drops them even though those flows do not occupy buffer space currently, BMRE can keep TCP flows from unresponsive flows even with small buffer sizes. It should be remarked that the average queue size per a flow is smaller than 0.25 packets when the number of flows is 650, which means majority of TCP flows are in retransmit timeouts state.

In Figure 4.6, we measure the instantaneous goodput of each flow to observe the instantaneous behavior of TCP and CBR flows. The number of TCP flows is set to 12 and the number of CBR flows is set to 8. All CBR flows send data at 3 Mbps which is three times as large as the fair share value. The measurement interval T_m is set to 0.5 seconds. Because RED can not



(a) When N is varied from 5 to 90



(b) When N is varied from 100 to 800

Figure 4.5: Average of ($Goodput_i/Fair\ Share$) vs. number of flows

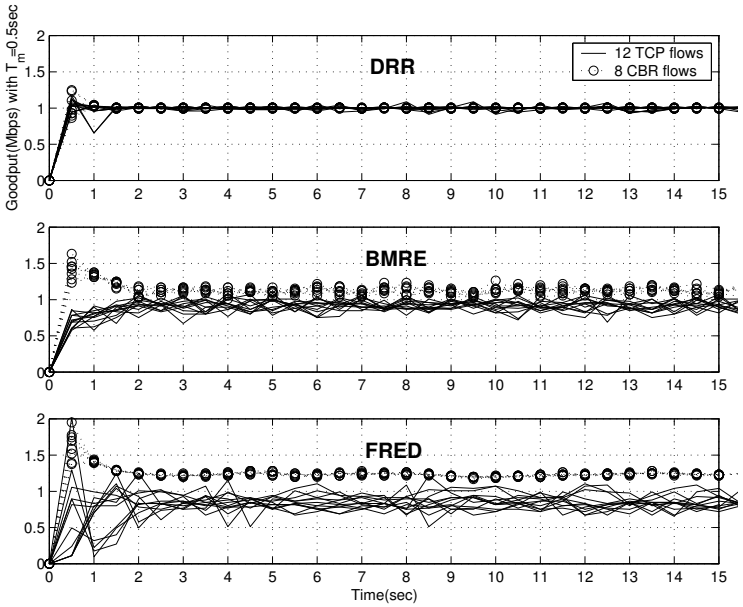


Figure 4.6: Instantaneous $Goodput_i$ in case of $T_m = 0.5$

protect TCP flows from unresponsive flows, we have excluded RED from this simulation. The instantaneous goodput of DRR is extremely smooth because there are only 20 flows and each flow can buffer a sufficient number of packets. The instantaneous goodput of TCP flows for BMRE is much smoother than that for FRED. BMRE regulates CBR flows within 2 seconds. When the number of flows is small, BMRE's operation is mainly based on both per-flow exponential adjustment of drop probability which drops packets nearly periodically, and a virtual threshold function that bounds both per-flow queue sizes and the global queue size.

4.3 Instantaneous Rates of TCP and TFRC Flows

We define Coefficient of Variation for TCP flow i as follows:

$$CoV_i = \sqrt{\frac{1}{T/T_m - 1} \sum_{k=0}^{T/T_m - 1} \left(\frac{Goodput_i(kT_m, kT_m + T_m)}{Goodput_i(0, T)} - 1 \right)^2}. \quad (4.4)$$

First, we simulate only TCP flows and results are shown in Figure 4.7 because many real-time applications still use TCP as their transport protocols. IP data packet size is set to 500 bytes because real-time applications would decrease its packet size to reduce transmission delay and to cope with bursty packet loss. BMRE maintains much smaller CoV_is than FRED and RED do. When there are small number of flows such that N is smaller than 60, BMRE outperforms FRED due to its periodic packet drops. Also, Fair Drop greatly improves the overall performance of BMRE when there are many flows such that N is larger than 180. DRR greatly outperforms three other schemes due to its per-flow queueing.

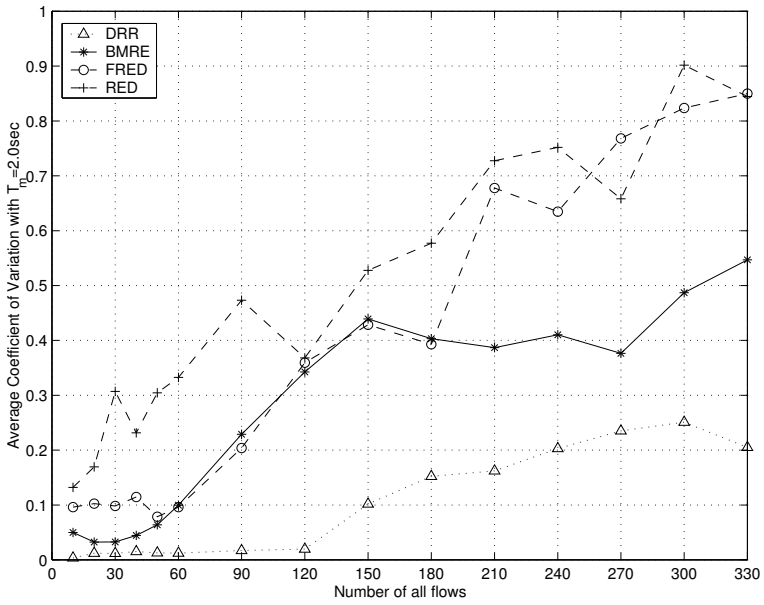


Figure 4.7: Average coefficient of variation for TCP flows vs. number of flows in case of $T_m = 2.0$

Secondly, we simulate 40% TFRC [13, 14] flows and 60% TCP flows and measure the mean CoV_is of TFRC flows. All parameters are set to values from [13]. TFRC operates with an equation-based rate control that characterizes

TCP sending rates [19, 24] based on the following equation:

$$SR = \frac{s}{R\sqrt{\frac{2p}{3}} + t_{RTO}(3\sqrt{\frac{3p}{8}})p(1+32p^2)}. \tag{4.5}$$

An upper bound on the sending rate SR is used, which is a function of the steady-state loss event rate p, data packet size s in bytes, round-trip time R, and TCP retransmit timeout value t_{RTO}. TFRC estimates the average loss interval, which is a weighted sum of last n loss intervals considering consecutive packet loss events as a single loss event. TFRC uses the average loss interval to calculate the sending rate. We can easily see that TFRC flows should experience periodic packet loss events to estimate p accurately without noisy fluctuation.

Results are shown in Figure 4.8. As can be seen from this figure, TFRC flows in RED experience noisy instantaneous goodputs, in contrast to BMRE and FRED. This feature of BMRE should encourage the adoption of TFRC

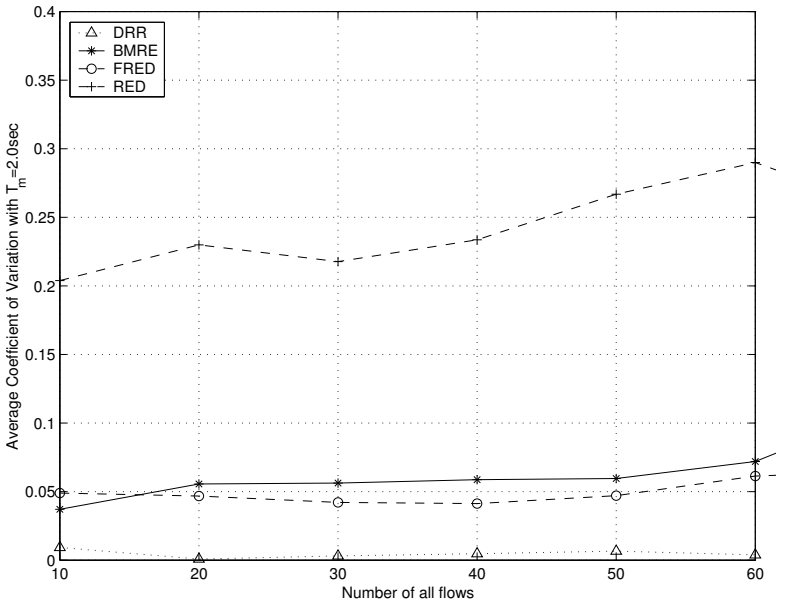


Figure 4.8: Average coefficient of variation for TFRC flows vs. number of flows in case of $T_m = 2.0$

for real-time applications as a congestion control mechanism.

It should be mentioned that TFRC algorithm is currently not well tuned because TCP models shown in [19, 24] is currently not exact. We found that TFRC receives different shares compared with TCP when the number of flows is large and the packet drop probability is high. Furthermore, (4.5) is derived with assumption that packet drop events are Bernoulli trials and that assumption is valid only for RED. This assumption is not valid in general as shown in [25]. It is needed to acquire more accurate TCP model. Also, we think that a slightly modified version of TCP that reduces TCP's coarse retransmit timeouts would satisfy real-time applications if BMRE is adopted as a buffer management scheme.

4.4 Throughput Differentiaion of Weighted BMRE

BMRE can be extended to support flows with different weights. To support differentiated shares, we add a new per-flow variable w_i (a weight value for flow i), and a portion of code is modified. We use two bits of the TOS (Type of Service) field in the IP header. To support weighted BMRE, N_{flow} indicates the total weights of active flows. The values of $rate_{fair}$ and max_{th} should be multiplied by w_i. We simulate 8 TCP flows with weights of 1, 2, 3 and 4. Results are shown in Figure 4.9. Although weighted BMRE can not differentiate per-flow queueing delays because it uses single FIFO buffer, weighted BMRE can be used to effectively support different per-flow shares of goodputs.

4.5 Miscellaneous Topics

4.5.1 Considerations for Implementation

Cooperation and negotiation among several ISPs would probably not be easy. Therefore, a buffer management algorithm should be able to operate individually. We can exploit the performance of BMRE without installation on several contiguous routers because BMRE routers operate individually without any exchange of additional information.

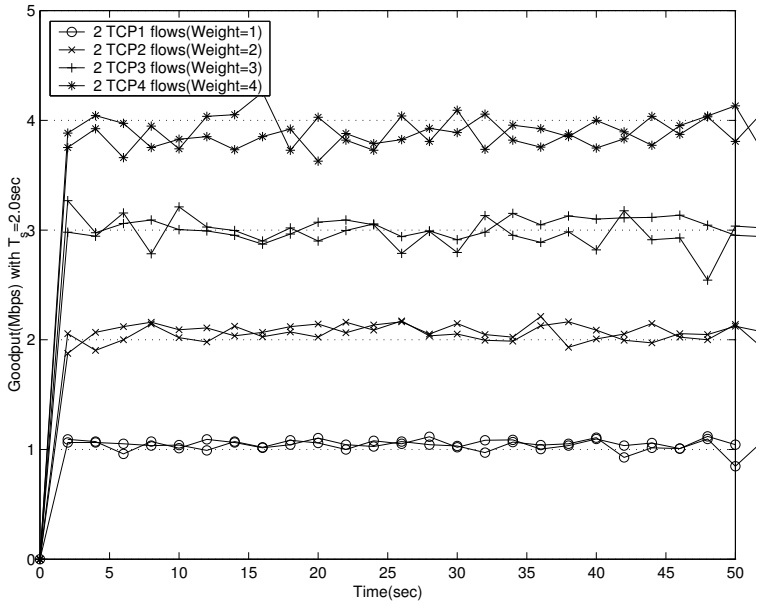


Figure 4.9: Instantaneous Goodput of weighted BMRE

4.5.2 Queueing Delay and IP Packet Size

Although we preset router buffer sizes in Section 3.6, how many flows a router would support should be considered first. If a router would support $30,000$ TCP flows with guaranteeing fairness for each TCP flow, router buffer size should be at least $30,000$ IP packets. Assuming that each IP packet size is 1 kbytes and each flow needs at least 2 packets buffer, a router buffer size should be at least 60 Mbytes. If a router supports an OC-12c link, maximum queueing delay would be 0.77 seconds, which is unacceptably high and flows transferring real-time application data would not be satisfied. This unacceptably high delay is mainly due to large IP packet sizes. To mitigate large queueing delays caused by large IP packet sizes, a buffer management scheme should be able to support many flows with a smaller buffer size. Perhaps, a major difficulty in next generation IP routers would be to reduce queueing delays. As shown in Section 4.2, BMRE could support about 350 flows limiting average queue size below 140 kbytes which means that each

flow buffers 0.4 packets in average.

4.5.3 Comparison of DRR and BMRE

The main advantage of BMRE is that it achieves comparable performance to DRR while BMRE maintains a single FIFO buffer. Although DRR achieves perfect fairness for flows that have currently at least one packet in the router buffer, it cannot ensure fairness when the number of flows increases above a certain threshold. From the simulation results of BMRE and DRR, we can see that BMRE achieves comparable performance when the number of flows is small and outperforms DRR when the number of flows is large. To allow DRR to outperform BMRE, the buffer size should be increased and queueing delay should be increased accordingly.

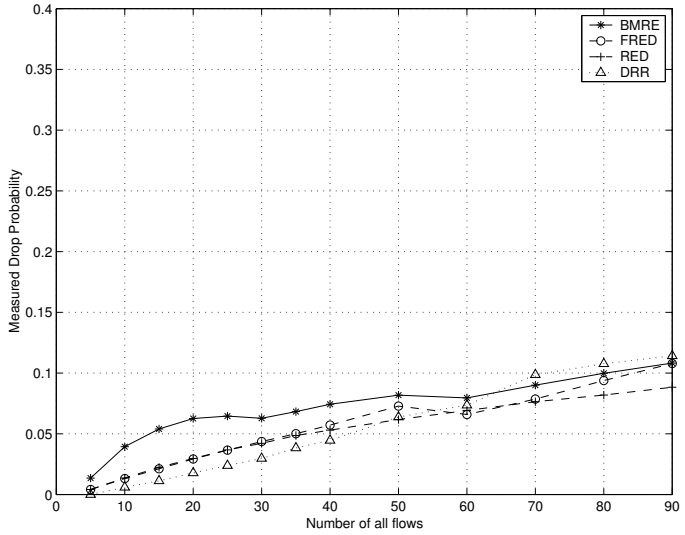
Furthermore, DRR uses per-flow queueing and per-flow scheduling that are considerably hard to implement and does not consider the large amount of legacy routers that use a single and simple FIFO buffer for its each output link. It would be hard to implement per-flow queueing and per-flow scheduling practically. Because DRR guarantees fairness only for currently backlogged flows, DRR would not be an optimum scheme for highly bursty flows such as TCP and its cost per performance ratio would be very high.

4.5.4 Frequent Packet Drop

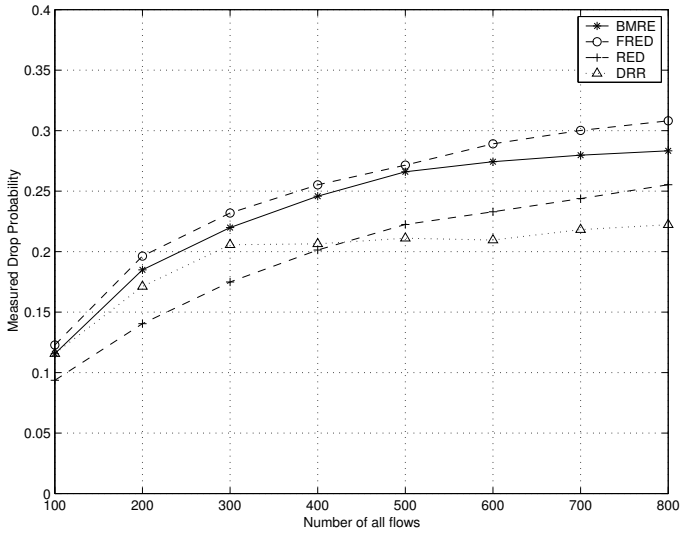
As shown in Figure 4.4, BMRE drops packets more frequently than RED, FRED and DRR. To see how many packets are dropped in each scheme, we measured and plotted packet drop probabilities in Figure 4.10(a) and 4.10(b) with the same configuration in Section 4.2.1.

Measured drop probability is defined as the ratio of number of dropped packets to number of packets sent by TCP sources. BMRE drops packets more frequently to guarantee fairness and reduce queueing delays. It should be reminded that there are trade-offs between reducing queueing delays and reducing packet drops in current IP networks. While BMRE maintains much shorter queueing delays when the number of flows are smaller than 50, BMRE drops packets much more frequently than other schemes in that region. More frequent packet drop results in the large fraction of wasted link capacity in the

previous links and is a main weak point of BMRE. This problem is mainly due to that TCP detects congestion only by packet drop and there is no additional information exchange about congestion between TCP and routers. But, we believe that this problem can be solved by using ECN (Explicit Congestion Notification) marking.



(a) When N is varied from 5 to 90



(b) When N is varied from 100 to 800

Figure 4.10: Measured drop probability vs. number of flows

5. Conclusions

We have proposed a dynamically adjusting per-flow buffer management scheme that can be applied to TCP flows and to flows transferring real-time application data. We have simulated various configurations with TCP, CBR and TFRC flows. BMRE exhibits better fairness, less delays, and better smoothness of sending rates than previous schemes. Introduction of a virtual threshold function that divides router operation into three modes allows the average queue size to fluctuate around the $target_q$ value and eliminates unnecessary delays. BMRE also produces more efficient buffer usage and helps routers support more flows than RED, FRED and DRR with the same buffer size. The per-flow rate estimation was accurate in view of estimating the per-flow current share, and noisy and rapid fluctuations were filtered. The per-flow exponential adjustment of the drop probability prevents unresponsive flows from achieving an unfairly large share. BMRE also controls the per-flow queue size, preventing flows from buffering more than a sufficient number of packets or buffering fewer than the necessary number of packets.

We also introduced a practical definition of "active flows" and developed a new algorithm for routers to support a larger number of flows in a fair manner in spite of insufficient buffer size. With practical definition of "active flows", BMRE reduces overhead coming from maintaining a large number of per-flow states. Fair Drop greatly improves overall performance when the number of flows are large and the average queue size of a flow in a router buffer is less than one packet. Moreover, the use of bits in the TOS field of the IP header allows easy differentiation of bandwidth allocation. We believe that BMRE can support real-time applications and can encourage the use of end-to-end congestion control mechanisms such as TFRC.

Although BMRE improves the overall performance of buffer management scheme, additional work to be done remains. Analysis on how TFRC and its

variants can better interoperate with BMRE, RED, and FRED is needed. Research on tuning the parameters and algorithms of TCP-Friendly Rate Control is needed to satisfy the requirements of real-time applications. More functions should be added to produce smoother sending rates for TCP-Friendly flows. Although BMRE achieves better fairness on multiple congested links than RED or FRED, research on optimization of BMRE and TCP is needed to achieve much better fairness on multiple congested links.

Appendix: A Detailed Pseudocode of BMRE Algorithm

In this Appendix, we present a detailed pseudocode of BMRE algorithm that was used for simulation.

- **Constants:**

$max_q = 320000;$	// maximum aggregate queue size (bytes)
$fd_{th} = 140000;$	// fair drop operates if q is larger than this
$max_{targetq} = 120000;$	// maximum value of $target_q$
$max_{nflow} = 320;$	// maximum N_{flow} that can be maintained
$\alpha = 1.2;$	// increase factor of β
$w_q = 0.004;$	// weight for average queue size calculation
$\beta_{max} = 10;$	// maximum β
$\beta_{min} = 0.5;$	// minimum β
$\beta_{init} = 7.5;$	// initial β
$K = 0.15sec;$	// constant used for rate estimation
$BW = 2500000Bps;$	// service rate (bytes per sec)
$Timeout\_Value = 0.45sec;$	// timeout value used for flow expiration

- **Global Variables:**

$N_{flow};$	// number of active flows (initially, 0)
$vmax_q;$	// virtual maximum buffer size (bytes)
$max_{th};$	// maximum queue size for each flow (bytes)
$target_q;$	// target queue size (bytes)
$rate_{fair};$	// fair rate(share)
$time;$	// current real time (sec)

- **Global Queue Sizes:**

$q;$	// current global queue size (bytes) (initially, 0)
$avg_q;$	// average global queue size (bytes) (initially, 0)

- **Per-flow Variables:**

$q[i]$; // queue size (bytes)
$rate[i]$; // estimated rate (bytes per sec)
$\beta[i]$; // $\beta[i]$
$count[i]$; // number of bytes processed since last $\beta[i]$ update
$qtime[i]$; // last time packet is buffered (sec)

- **Functions:**

find_i(p); // find the flow number to which p belongs
update_gv($mode$) { // update global variables
    if ($mode == 1$) $N_{flow} = N_{flow} + 1$;
    else $N_{flow} = N_{flow} - 1$;
    $vmax_q = \mathbf{vth}(N_{flow})$;
    $max_{th} = vmax_q / N_{flow}$;
    $target_q = \mathbf{min}(vmax_q/2, max_{targetq})$;
    $rate_{fair} = BW/N_{flow}$;
}
initialize_pfv(i) { // initialize per-flow variables
    $q[i] = count[i] = p.size$;
    $rate[i] = 0$;
    $\beta[i] = \beta_{init}$;
    $qtime[i] = time$;
}
update_gq($mode$) { // update q and avg_q
    if ($mode == 1$) $value = p.size$;
    else $value = -p.size$;
    $q = q + value$;
    $avg_q = (1 - w_q) \times avg_q + w_q \times q$;
}
update_beta($rate_{ratio}$) { // update $\beta[i]$
    if ($rate_{ratio} > 1$ && $avg_q > target_q$) {
        $\beta[i] = \beta[i]/rate_{ratio}$;
        if ($\beta[i] < \beta_{min}$) $\beta[i] = \beta_{min}$;
    } **else** {
        $\beta[i] = \beta[i] \times \alpha$;
        if ($\beta[i] > \beta_{max}$) $\beta[i] = \beta_{max}$;
    }
}
random(); // uniform random number in [0...1]
pow(a, b); // calculate and return a^b

exp(c); // calculate and return e^c

- **For each arriving packet p:**

```
1:  if ( q ≥ vmax_q || q ≥ max_q ) {
2:      drop(p);
3:      return;
4:  }
5:  if ( find_i(p) == false ) {
6:      if ( N_flow < max_nflow ) {
7:          i = new flow number;
8:          update_gv(1);
9:      } else {
10:         i = randomly selected from N_flow flows whose q[i] are 0;
11:     }
12:     initialize_pfv(i);
13:     update_gq(1);
14:     return;
15: }
16: if (q ≥ fd_th ) {
17:     drop(p);
18:     return;
19: }
20: q[i] = q[i] + p.size;
21: count[i] = count[i] + p.size;
22: if ( count[i] ≥ 4 × max_th ) update_beta(rate[i]/rate_fair);
23: u = random();
24: if ( u < pow(q[i]/max_th, β[i]) ) {
25:     q[i] = q[i] − p.size;
26:     drop(p);
27: } else {
28:     update_gq(1);
29:     dt = time − qtime[i];
30:     qtime[i] = time;
31:     rate[i] = (1 − exp(−dt/K)) × p.size/dt
            + exp(−dt/K) × rate[i];
32: }
```

- **For each departing packet p:**

33: **find_i**(p);
34: $q[i] = q[i] - p.size$;
35: **if** ($count[i] \geq 4 \times max_{th}$) **update_beta**($rate[i]/rate_{fair}$);
36: **update_gq**(0);

- **For each flow expiration:**

  //each flow expires **if** ($time - qtime[i] \geq Timeout\_Value$)
  37: **update_gv**(0);

요 약 문

IP 망에서의 전송율 측정에 기반한 동적 버퍼 관리

본 논문에서는 라우터 버퍼에서의 패킷의 대기 시간을 줄이고 또한 많은 수의 TCP 연결들을 지원하는 방안을 제안한다. 현재 인터넷에서의 트래픽의 양은 늘어나는 반면 대부분의 라우터들은 이러한 트래픽들에 대해 공평성을 보장하거나 서비스 품질을 보장하기 위한 방안을 거의 사용하고 있지 않다. 또한, 실시간성을 요구하는 트래픽들은 이렇게 기본적인 공평성이나 서비스 품질을 만족하지 못하는 네트웍 상황에서는 TCP와 호환성이 있는 혼잡 제어 방식을 사용하려하지 않는다.

따라서, 본 논문에서는 인터넷에서의 기본적인 공평성과 서비스 품질을 보장하는 방안을 제안한다. 제안하는 동적 버퍼 관리 방안은 가상 문턱 함수와, 정밀하고 안정한 접속별 전송율 측정과, 접속별 지수함수적인 패킷 폐기 방법과, 많은 접속이 있을 때에 공평성을 보장하는 방법을 포함한다. 또한, “접속”의 의미를 좀 더 실질적으로 정의하여 접속별 정보를 저장하는 비용을 줄이고자 한다.

세 가지의 영역으로 나뉘어서 동작하는 가장 문턱 함수를 사용하여 접속별 대기열 크기를 조절한 같은 크기의 버퍼를 사용하는 경우에 기존에 제안된 RED, FRED, DRR에 비해 더 좋은 성능을 얻을 수 있었으며 많은 접속이 있을 경우에 공평성을 보장하는 방안은 하나의 접속이 평균적으로 하나의 패킷도 라우터의 버퍼 내에 버퍼링 할 수 없는 상황에서도 공평성을 보장할 수 있다.

References

[1] D. Tan, A. Zakhor, "Real-time Internet Video Using Error Resilient Scalable Compression and TCP-friendly Transport Protocol," *IEEE Trans. Mutimedia*, vol. 1, no. 2, pp. 172–186, June 1999.

[2] A. Mena, J. Heidemann, "An Empirical Study of Real Audio Traffic, *Proc. of IEEE INFOCOM2000*," pp. 101–110, March 2000.

[3] A. Demers, S. Keshav, S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," *Proc. of ACM SIGCOMM'89*, pp. 1–12, September 1989.

[4] P.E. McKenney, "Stochastic Fairness Queueing," *Proc. of IEEE INFOCOM'90*, pp. 733–740, June 1990.

[5] M. Shreedhar, G. Varghese, "Efficient Fair Queueing using Deficit Round Robin," *Proc. of ACM SIGCOMM'95*, pp. 231–242, September 1995.

[6] I. Stoica, S. Shenker, H. Zhang, "Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks," *Proc. of ACM SIGCOMM'98*, pp. 118–130, September 1998.

[7] S. Floyd, V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Trans. Networking*, vol. 1, no. 4, pp. 397–413, August 1993.

[8] D. Lin, R. Morris, "Dynamics of Random Early Detection," *Proc. of ACM SIGCOMM'97*, pp. 127–137, October 1997.

[9] S. Floyd, K. Fall, "Promoting the Use of End-to-end Congestion Control in the Internet," *IEEE/ACM Trans. Networking*, vol. 7, no. 4, pp. 458–472, August 1999.

[10] W. Feng, D.D. Kandlur, D. Saha, K.G. Shin, "A Self-Configuring RED Gateway," *Proc. of IEEE INFOCOM'99*, pp. 1320–1328, March 1999.

[11] R. Mahajan, S. Floyd, "Controlling High-Bandwidth Flows at the Congested Router," Work in progress, November 2000, URL http://www.aciri.org/red-pd/.

[12] S. Floyd, K. Fall, "Router Mechanisms to Support End-to-End Congestion Control," LBL Technical Report, February 1997.

[13] S. Floyd, J. Padhye, J. Widmer, "Equation-Based Congestion Control for Unicast Applications," *Proc. of ACM SIGCOMM2000*, pp. 43–56, September 2000.

[14] S. Floyd, J. Padhye, J. Widmer, "TFRC, Equation-Based Congestion Control for Unicast Applications: Simulation Scripts and Experimental Code," February 2000, URL http://www.aciri.org/tfrc/.

[15] R. Rejaie, M. Handley, D. Estrin, "RAP: An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet," *Proc. of IEEE INFOCOM'99*, pp. 1337–1345, March 1999.

[16] D. Sisalem, H. Schulzrinne, "The Loss-Delay Based Adjustment Algorithm: A TCP-Friendly Adaptation Scheme," *Proc. of NOSSDAV'98*, 1998.

[17] R. Morris, "TCP Behavior with Many Flows," *Proc. of IEEE ICNP'97*, October 1997.

[18] B. Suter, T.V. Lakshman, D. Stiliadis, A.K. Choudhury, "Buffer Management Schemes For Supporting TCP in Gigabit Routers With Per-flow Queueing," *IEEE J. on Selected Areas in Commun.*, vol. 17, no. 6, pp. 1159–1169, June 1999.

[19] J. Padhye, V. Firoiu, D.F. Towsley, "Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation," *IEEE/ACM Trans. Networking*, vol. 8, no. 2, pp. 133–145, April 2000.

[20] V.P. Kumar, T.V. Lakshman, D. Stiliadis, "Beyond Best Effort: Router Architectures for the Differentiated Services of Tomorrow's Internet," *IEEE Comm. Mag.*, vol. 36, no. 5, pp. 152–164, May 1998.

[21] J. Padhye, S. Floyd, "Identifying the TCP Behavior of Web Servers," ICSI Technical Report TR-01-002, February 2001, URL http://www.aciri.org/tbit/.

[22] K. Fall, S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 26, no. 3, pp. 5–21, July 1996.

[23] UCL/LBNL/VINT Network Simulator - ns (version2), URL http://www.isi.edu/nsnam/ns/.

[24] M. Mathis, J. Semke, J. Mahdavi, "The Macroscopic Behavior of TCP Congestion Avoidance Algorithm," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 3, pp. 25–41, July 1997.

[25] M. Mitzenmacher, R. Rajaraman, "Towards More Complete Models of TCP Latency and Throughput," *The Journal of Supercomputing*, vol. 20, no. 2, pp. 137–160, September 2001.

감사의 글

지난 2년 동안 석사과정 생활을 하면서 여러가지 힘든 일들이 많았지만 주위에 많은 분들이 도움을 주신 덕분에 이렇게 작은 결실을 맺게 되었습니다. 그러한 분들께 이 자리를 빌어 감사의 말씀을 전하고자 합니다. 항상 최선을 다해 부족한 저를 이끌어 주시고 논문을 지도해 주신 조동호 교수님께 깊은 감사를 드립니다. 또한, 부족한 제 논문을 심사해 주시고 좋은 충고를 해 주신 성단근 교수님과 이황수 교수님께도 감사드립니다.

실험실 생활을 하는 동안 여러 사람들을 만나고 다양한 경험을 할 수 있어서 많은 도움이 되었고 정신적으로 성숙할 수 있는 계기가 되었던 것 같습니다.

항상 좋은 충고 아끼지 않고 격려해 주시던 성원이 형, 그리고 언제나 웃는 얼굴로 대해 주신 동준이 형, 항상 열심히 연구하시던 범식이 형, 재밌게 사시는 모습이 보기 좋았던 성홍이 형에게 감사드립니다. 그리고, 언제나 따뜻한 마음으로 대해 주시고 고민 상담을 많이 해 주신 종욱이 형과 승식이 형, 같이 담배 피며 인생사를 논하던 일이 형, 재치가 뛰어난 중희 형, 같이 프로젝트 하면서 많은 것들을 신경 써 주셔서 너무나 감사한 영욱이 형, 커피와 음료수를 즐기며 많은 이야기를 했던 규태 형, 항상 다정하게 대해 주신 재우 형, 자주 보지 못해 아쉬운 득형이 형, 그리고 다정한 경상도 사나이였던 경호 형, 편안하게 대해 주시고 마음 써 주신 선호 형, 도움 받은게 너무 많아 갚을 수가 없는 성관이 형, 항상 보고 싶은 휴대 형, 항상 열심히 하시던 은정이 누나, 연구실을 밝게 만들었던 주민이 누나에게 감사드립니다.

예쁘고 착하고 다정한 지영이 누나와 혜정이에게도 감사드립니다. 그리고, 여러 가지 일들을 맡아 고생 많았던 후배들에게도 감사드립니다. 항상 열심히 하고 예의바른 태수 후배님, 부지런하고 잘 생긴 현호 후배님, 잡일을 도맡아 하던 기호 후배님, 착하고 부지런한 유철 후배님에게 감사드립니다. 이분들의 앞날이 창창하시길 기원합니다. 또한, 5층 휴게실에서 인생사를 이야기하던 재훈이 형, 성호 형, 호형이에게도 감사드립니다. 지면이 모자라 수식이 부족한 점 죄송합니다.

그리고, 제 몸을 걱정하며 항상 신경 써 주신 부모님과 사랑하는 동생에게도 감사드립니다.

이 력 서

성　　명 : 조 정 우 (曺 政 佑)

생년월일 : 1978년 9월 20일

출 생 지 : 경상남도 거제시

본　　적 : 경상남도 거제시

학 력

1996.3 – 2000.2 : 한국과학기술원 전기 및 전자공학과(B.S.)

2000.3 – 2002.2 : 한국과학기술원 전자전산학과(M.S.)

학 회 활 동

1. Jeong-woo Cho and Dong-ho Cho, "A Dynamic Buffer Management Scheme Based on Rate Estimation in Packet-Switched Networks," *IEEE GLOBECOM2001*, pp. 2304-2310, 2001.
2. Jeong-woo Cho, Young-uk Chung, Changhoi Koo, DS Park, Daegyun Kim and Dong-ho Cho, "Multiple Quality Control: A New Framework for QoS Control in Forward Link of 1xEV-DV Systems," *IEEE VTC2002 Spring*, Accepted for Publication.