# Rate And Queue Controlled Random Drop (RQRD): A Buffer Management Scheme for Internet Routers

Mansour J. Karam, Fouad A. Tobagi

*Abstract*—**A buffer management scheme destined to be deployed in Internet routers should provide an adequate handling of all traffic types, including UDP and TCP traffic as well as fairness among competing flows. In core routers, keeping per-flow state and taking per-flow actions is strongly discouraged, because of the scalability issues that ensue. Consequently, CSFQ [12] adopts a distributed architecture, and uses rate information to provide fairness in core routers. However, we show that CSFQ is sensitive to the setting of its parameters and often leads to a reduction in aggregate throughput. In this paper, we present *Rate and Queue controlled Random Drop* (RQRD), a buffer management scheme which is based on CSFQ's distributed architecture, but adds queue size information, as in RED, and provides two drop precedences to achieve all the objectives above. We show that RQRD provides an adequate service in the context of both the present and the future Internet.**

## I. Introduction

The Internet must support traffic types which are very different in nature, generated by various applications using either the TCP or UDP transport protocol. In today's Internet, traffic of all types is served by a single FIFO queue and share the same bandwidth resources on the link. The traditional view which is that TCP traffic is the real and useful traffic, whereas UDP traffic is only an additional nuisance[1] has led to the development of buffer management schemes either aimed to serve TCP traffic, or to protect TCP traffic from competing UDP flows (e.g., RED [4], CSFQ [12]). However, in the context of an Integrated Services network, UDP is not always undesirable, despite its non-responsive nature. (For example, in the case of a video stream, the excessive loss of packets renders the stream useless at the receiver.) Consequently, (1) it is important for a buffer management scheme to provide a service that is adequate to both UDP and TCP traffic. In addition, it should be able to differentiate between different UDP flows, in support of applications such as layered video and voice. Moreover, (2) a buffer management scheme should be equipped to provide fairness among competing flows. For example, unfriendly UDP traffic should not starve competing TCP flows. Also, if different customers are paying for the same service, each should be capable of getting the same service rate from the network. Nevertheless, no per-flow state should be kept in routers given the scalability issues that ensue. Also, fairness should not be achieved at the expense of throughput. Finally, (3) a buffer management scheme should be robust. It should perform effectively for a wide range of conditions in terms of buffer size, traffic characteristics and traffic mix. Also, its performance should not be too sensitive to the setting of its parameters.

The authors are with the Computer Systems Laboratory, Stanford University, Stanford CA 94305 USA (e-mail: {mans, tobagi}@stanford.edu).

[1]This belief is triggered by the original vocation of the Internet, which is to serve TCP data traffic, and is reinforced by the responsive nature of TCP traffic.

Since keeping per-flow state and taking per-flow actions is strongly discouraged, the design of such a scheme is a challenging task. For example, RED uses queue size information to improve TCP throughput but is neither equipped to provide fairness among various competing flows, nor is effective in the presence of UDP traffic; on the other hand, CSFQ uses rate information to protect TCP traffic from competing UDP flows, but (as we show in this paper) can lead to a massive reduction in achievable throughput.

In this paper, we present *Rate and Queue controlled Random Drop* (RQRD). Based on CSFQ's distributed architecture, RQRD takes into account both rate (just as CSFQ) and queue size (just as RED) information to achieve the objectives described above. We start in Section II with an overview of prior work. In Section III, we state our assumptions and describe the details of the RQRD algorithm. In Section IV, we use simulation to compare RQRD to existing queue management schemes. Finally, Section V summarizes the results of this paper. A longer version of this paper including the algorithm pseudocode can be found at http://www-mmnetworks.stanford.edu.

## II. Prior Work

According to Drop Tail (DT [5]) which, owing to its simplicity, is used in most routers today, incoming packets that find the queue full are simply dropped. Yet, DT suffers from a number of shortcomings, preventing it from acting effectively in the presence of congestion: first, there is no way of ensuring fairness among flows traversing a Drop Tail queue; second, in the case of TCP traffic, DT queues have been shown to introduce *global synchronization* in the network, leading to both a decrease in the average throughput and an increase in the average delay through the link [5]. To avoid DT's shortcomings, RED [4] attempts to avoid buffer overflow by controlling the queue size so it is kept at a reasonably low level. Alternatively, CSFQ [12] attempts to reduce the effect of congestion on individual flows by allocating to each flow a minimum portion of the resources, independently of other traffic in the network. In this section, we present a description of RED and CSFQ.

### A. Random Early Detection (RED)

RED [4] is a mechanism that attempts to provide a high aggregate throughput while keeping the queue size small by dropping packets probabilistically *before* the buffer overflows, using for that purpose a probability drop function that increases with the average queue size[2]. Designed for TCP traffic, the assump-

[2]The average queue size itself is computed using an *Exponential Weighted Moving Average* (EWMA) when the queue is non-empty [4]. (In case the queue

tion is that marking a packet causes the source to reduce its rate of transmission (by reducing the TCP window size). [4] shows that by getting rid of global synchronization, RED is capable of reducing the average queue size, while increasing the average throughput on the link. However, RED is shown to be sensitive to the setting of its parameters ([3], [7]). [3] also finds that unless the buffer size is large enough, RED is incapable of avoiding buffer overflow. In addition, [7] finds that in some situations RED does not bear any progress in terms of fairness when compared to DT. Finally, RED is not equipped to insure fairness among a number of streams flowing through the buffer; in particular, in case UDP and TCP traffic are mixed together, TCP gets a significantly lower portion of the link bandwidth.

In light of RED's shortcomings, other approaches have been investigated through the development of competing schemes, e.g., BLUE [3]. Also, a number of buffer management schemes that are based on RED have been developed, in order to alleviate the lack of fairness that results from the basic RED algorithm, e.g. RIO ([2], [10], [6], [11]) and CHOKe [8].

### *B. Core-Stateless Fair Queuing (CSFQ)*

Similarly to RED, CSFQ [12] assumes that routers are equipped with FIFO queues. As in RED, admittance to the queue is controlled by an algorithm which computes a drop probability for each packet that arrives to the queue. However, whereas RED uses the queue size as a control parameter, CSFQ uses rate information in an attempt to insure fairness among the flows sharing the buffer. The Internet core is assumed to be partitioned into different administrative domains, called core networks. In order to avoid keeping per-flow information and performing per-flow actions in core routers, CSFQ is a distributed algorithm, where edge and core routers[3] exercise different, complementary functions.

*Edge Router Function.* Edge routers keep a table of all flows entering the core network; they estimate the rate of each flow and insert this estimate as a label in the packet. More specifically, upon the arrival of a packet belonging to flow *i* at time *t*, the current rate of that flow is estimated using $r_i^{new} = (1 - e^{-T_i/K})\frac{l_i}{T_i} + e^{-T_i/K} r_i^{old}$,where $r_i^{old}$ denotes the old estimate of the rate of the flow, $T_i$ represents the current inter-arrival time between the previous and current packets pertaining to flow $i$ and $l_i$ is the size of the current packet. The result is a moving exponential average of the rate $r_i$ of flow $i$[4]. The packet is then stamped with $r_i$ and transmitted into the core network.

*Core Router Function.* Core routers decide on whether to drop a packet based on aggregate measurements and the label that is inserted in the packet. More specifically, the core router computes the aggregate arrival rate to the queue $A(t)$, the rate of packets admitted to the queue $F(t)$, and uses both $A(t)$ and $F(t)$ to estimate the fair share of the link bandwidth $\alpha(t)$ for each flow that contends for the resources. The drop probability is then computed as $p_R = \max(0, 1 - \frac{\alpha(t)}{r(t)})$. $A(t)$ and $F(t)$ are computed using the same exponential averaging method used by the edge router for $r(t)$.

Denoting by $C$ the total link bandwidth, the fair share estimation algorithm is as follows. A congestion bit $c_l$ is defined, which value depends on $A(t)$ and $C$. More specifically, $c_l$ is initialized to 0. If $A(t)$ exceeds $C$ for a time period exceeding $K_v$, $c_l$ is set to 1, indicating that packets must be dropped in order to prevent buffer overflow. Conversely, If $A(t)$ is less than $C$ for a time period exceeding a window $K_v$, $c_l$ is reset to 0. The fair share $\alpha(t)$ is updated once each $K_v$ period. In case $c_l = 0$ at time *t*, $\alpha(t)$ is simply set to the largest per-flow rate seen at the input of the queue during the period $[t - K_v,\ t]$. In case $c_l = 1$, then $\alpha(t)$ is updated using $\alpha_{new} = \alpha_{old} \cdot \frac{C}{F}$. Note that updating $\alpha(t)$ has the desired effect of adjusting $F(t)$ such that $F(t) \cong C$[5].

*Additional Considerations.* The rate estimate computed by the edge router and inserted in a packet header is only a good estimate of the rate of the flow to which the packet belongs as long as no packets are dropped inside the network. To alleviate this problem, each core router along the path of a packet modifies $r(t)$ using $r_{new} = \min(r_{old},\ \alpha)$. Also, CSFQ can be easily extended to provide competing flows unequal shares of the link. For this purpose, each flow $i$ is assigned a weight $w_i$. The Ingress router stamps each packet with $\frac{r_i}{w_i}$ instead of $r_i$. In addition, the expression for $p_R$ becomes $p_R = \max(0, 1 - \alpha\frac{w_i}{r_i})$.

*Goals and Shortcomings.* As its name indicates, the purpose of CSFQ is to prevent congestion by giving to each flow a fair share of the resources. In particular, simulations in [12] show that CSFQ is capable of protecting TCP flows from overwhelming UDP flows. In addition, contrary to traditional fair queuing schemes that keep per-flow state in routers, the advantage of CSFQ lies in it's simplicity and scalability: FIFO queues are maintained in core routers, which neither carry state nor perform actions on a per-flow basis.

However, we will show in our simulations that CSFQ suffers from three shortcomings that hinder its performance. First, CSFQ is not capable of protecting UDP traffic against TCP traffic. Second, we show in Section IV that in its attempt to provide fairness among competing flows, RQRD leads to a massive reduction in throughput in a number of common and realistic situations, in particular when applied to TCP traffic. Finally, CSFQ drops packets according to the drop probability $p_R$ as soon as $c_l$ is set to 1 even though there could be enough buffer space to accommodate the excess traffic. As suggested in [12], this makes the choice of the value for the *K* parameter essential to the operation of CSFQ[6].

is idle, then a different formula is used, so the queue average size decreases exponentially with the queue idle time [4].)

[3] Routers sitting at the edge of the domain, and those sitting inside the domain are called *edge* and *core* routers, respectively.

[4] As described in [12], the weight used ($e^{-T_i/K}$) results in a rate estimate that converges asymptotically to the real rate.

[5] As mentioned in [12], CSFQ includes two additional heuristics: first, CSFQ reduces $\alpha(t)$ by 1% each time the buffer overflows. Conversely, when $c_l$ is reset to 0, $c_l$ remains equal to 0 until the queue size exceeds half the buffer size. This feature insures that no packet is dropped when the buffer is almost empty ([12]).

[6] In fact, the amount of transient burstiness allowed by the estimation algorithm should be proportional to buffer size; thus, *K* must be proportional to the buffer size, i.e. $K = \gamma\frac{Q_l}{C}$. Also, the optimal value of $\gamma$ depends on the expected traffic burstiness, so *K* cannot always be set optimally in a realistic environment, where traffic burstiness constitutes a dynamic phenomenon. In fact, simulation will shed some light on the undesirable dependence of achievable throughput on *K* in case CSFQ is used. (See Section IV.)

## III. RQRD ALGORITHM

RQRD is based on the distributed architecture used in CSFQ. RQRD provides two drop precedences to differentiate loss sensitive traffic, such as voice and video traffic from other traffic in the network. Edge routers implementing RQRD keep for each flow[7] $i$, in addition to the estimate the rate of the flow $r_i$ the drop precedence of that flow, $s_i$. We assume two priority levels for dropping (1 designating the highest level, 0 the lowest). Flows that do not tolerate any packet drop belong to level 1 (e.g., the base layer of an UDP video stream). Flows that tolerate packet drop belong to level 0, and can include both TCP and UDP traffic. $r_i$ is estimated just as in CSFQ. (See Section II-B.) Each packet is stamped with both $r_i$ and $s_i$ prior to its transmission into the transit network.

An RQRD core router delays any dropping action until both the queue size increases significantly *and* the aggregate rate exceeds the total link bandwidth, indicating that the transient congestion experienced will most likely become persistent. In order to achieve this goal, RQRD uses two drop probability functions, a rate dependent $p_R$ and a queue size dependent $p_Q$.

*Drop Probability* $p_R$. $p_R$ relates to the fair share, as estimated by a core router $\alpha\,(t)$, $r\,(t)$ and $s$. In case $s = 1$, $p_R = 0$, meaning that the packet is only dropped in case of buffer overflow. In case $s = 0$, the drop probability $p_R$ is function of $r(t)$ and $\alpha(t)$, exactly as in CSFQ, using the formula shown in Section II-B.

The fair share estimation performed in core routers is done in a manner that is very similar to CSFQ, except that the drop precedence of incoming packets is also taken into account. In particular, an RQRD core router estimates the total rate of packets of drop priority 0 at the queue input, $A\,(t)$, and the total rate of packets of priority 0 and 1 admitted to the queue, $F\,(t)$ and $P\,(t)$, respectively; for that, it uses the same exponential averaging method described in Section II-B. The value of a link congestion bit, $c_l$ is obtained as in CSFQ (Section II-B), except that $C$ is replaced here with $C'\,(t) = C - P\,(t)$, which represents the service rate for packets of drop priority 0. The fair share $\alpha\,(t)$ is then updated exactly like in CSFQ.

*Queue Dependant Drop Probability* $p_Q$. In addition, RQRD uses a queue dependent drop probability function $p_Q$, which, as in RED, is a non-decreasing function of the average queue size $Q\,(t)$. The value of $p_Q$ is obtained as follows: as in RED, we define $Q_{min}$ and $Q_{max}$ to be two thresholds to which $Q\,(t)$ is compared, and $Q_{tot}$ to be the total buffer size. We also define a buffer congestion bit, $c_q$, and set its value according to the position of $Q\,(t)$ with respect to $Q_{min}$, $Q_{max}$ and $Q_{tot}$. More specifically, $c_q$ is initialized to 0. If $Q\,(t)$ exceeds $Q_{max}$ for a time period greater than a given time window $K_w$, then $c_q$ is set to 1. $c_q$ is only reset to 0 when $Q\,(t)$ drops back below $Q_{min}$ and stays under that value for a time period exceeding $K_w$. $p_Q$ is always set to 0 when $Q \leq Q_{min}$ and to 1 when $Q \geq Q_{max}$. For $Q_{min} \leq Q \leq Q_{max}$, $p_Q(Q)$ depends on the value of $c_q$. (See Figure 1.) If $c_q = 0$, then the algorithm uses an optimistic drop function $p_Q(Q) = p_{low}(Q)$ that increases slowly with $Q$. Conversely, when $c_q = 1$, then RQRD uses a more aggressive drop function. Therefore, unlike RED $p_Q$ is hysteretic in form in the region $[Q_{min},\, Q_{max}]$. Hysteresis is needed to deal with both UDP and TCP traffic. In case of TCP traffic, the drop of a packet leads to a relatively fast response from the source, leading to a fast decrease in the queue size. Hence, when RQRD is applied to TCP traffic, the congestion bit $c_q$ is rarely set to 1, and $p_Q\,(Q) = p_{low}\,(Q)$ most of the time, leading to a drop probability that depends on the queue size in a manner that is very similar to RED. However, in case RQRD is applied to unresponsive UDP traffic, the rate of arrival to the queue remains high, independently of the rate of packets dropped. In these conditions, the queue size can exceed $Q_{max}$ for a period exceeding $K_w$. $c_q$ is thus set to 1, and $p_Q\,(Q)$ to $p_{high}\,(Q)$, which in turn leads to a much more aggressive packet drop function. In Section IV, we demonstrate the effectiveness of such a hysteresis function in providing an adequate handling of both UDP and TCP traffic. One candidate $(p_{low},\, p_{high})$ pair, which we call *Ellipse*, is obtained by using symmetric elliptic functions. We have also experimented with other $(p_{low},\, p_{high})$ pairs, such as the *ON/OFF* and *RED* pairs[8]. (See Figure 1.)

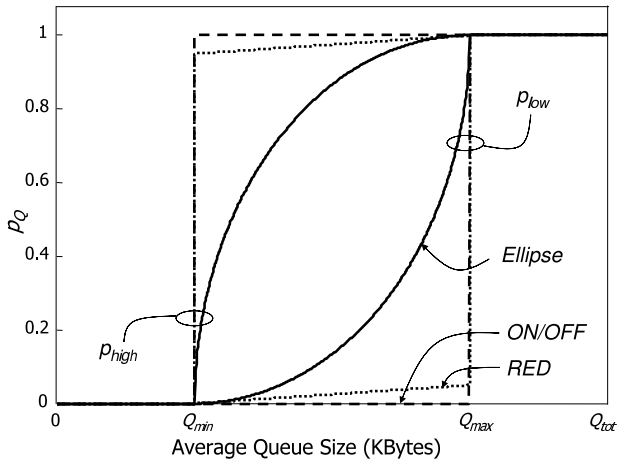


Fig. 1. Examples of $(p_{low},\, p_{high})$ pairs

The average queue size $Q\,(t)$ is found using the exponential averaging method described in II-A. (As done in RED [4], we use a different equation to update our estimate of $Q(t)$ in case the queue is idle.)

*Total Drop Probability* $p_d$. The drop probability function used by RQRD, $p_d$ consists of the product of both $p_Q$ and $p_R$, i.e. $p_d = p_R.p_Q$. Using such a drop probability function ensures that a packet is only dropped in case *both* the average queue size is large, and the rate of the flow the packet belongs to exceeds its fair share, meaning that the flow in question is most probably responsible for the congestion experienced on the link.

If either $c_q$ or $c_l$ is 0, RQRD assumes that the congestion is transient and that dropping packets is unnecessary, unlike CSFQ, which drops packets according to $p_R$ as soon as $c_l$ is set to 1 even though there could be enough buffer space to accommodate the excess traffic. Even though RQRD also uses $K$ in its estimation algorithms, the resulting performance proves to be much less sensitive to its specific value.

*Additional considerations.* In RQRD, a packet that arrives

[7] As in CSFQ, the operation of RQRD is independent of what is considered a flow. Hence, we intentionally keep the definition of the world *flow* imprecise.

[8] Simulation results have shown that the continuity of the curve is essential to the stability of the algorithm (see Section IV-A). For this reason, we use the *Ellipse* pair in most of our experiments.

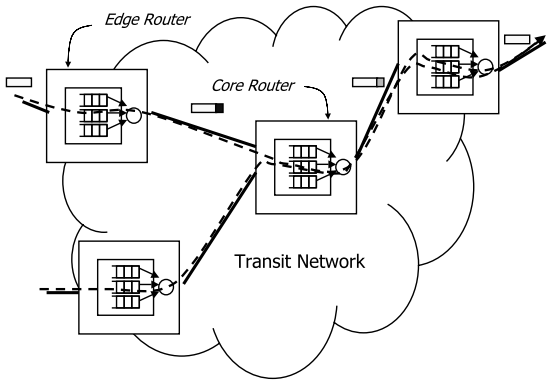


Fig. 2. Network scenario.

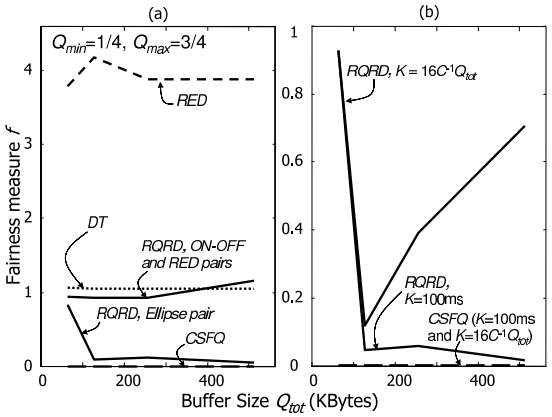


Fig. 3. $f$ versus $Q_{tot}$ for DT, RED, CSFQ and different RQRD implementations.

at the input of the queue at time *t* is relabeled using $r_{out} = [1 - p_d(t)]\, r_{in}$, where $r_{in}$ and $r_{out}$ constitute the original and new labels, respectively, and $p_d(t)$ represents the drop probability associated to the incoming packet. Also, extending the algorithm to support heterogeneous rates is done exactly as in CSFQ. (See Section II-B.)

## IV. Simulation Results

In this section, we conduct simulation experiments that shed a light on the potential benefits of RQRD. The general scenario (see Figure 2) consists of flows entering the core network through a number of edge routers, and then forwarded to a core router. We are interested in per-flow average throughput measurements at the output of the queue. We use $C$ = 10Mb/s, and experiment with values for the total buffer size $Q_{tot}$ varying from 64 to 512KBytes. We experiment with different round trip times by considering propagation delays $\tau$ ranging from one to 20ms. (Unless stated otherwise, we use a propagation delay of 1ms, that is a round trip time of 2ms.) In the RQRD algorithm, we use the same value for all constants pertaining to the estimation of the different parameters (that is, *K*, $K_A$, $K_F$, $K_Q$, $K_P$, $K_v$ and $K_w$). As for $Q_{min}$ and $Q_{max}$, the default chosen values are $\frac{1}{4}Q_{tot}$ and $\frac{3}{4}Q_{tot}$. However, we experiment with other $(Q_{min},\ Q_{max})$ combinations to understand their effect on the algorithm. For RED, we use $min_{th} = 0.3Q_{tot}$, $max_{th} = Q_{tot}$, $w_q = 0.1$ and $max_p = \frac{1}{50}$.

Since RQRD is designed to work well with both UDP and TCP traffic, we consider both traffic types in our experiments. UDP traffic consists of a succession of bursts of size *M*. The burst inter-arrival time *T* has a mean $T_f$ and is uniformly distributed (in the range $[0,\ 2T_f]$). In the case of Constant Bit Rate (CBR) UDP traffic, we use a constant burst size $M = M_f$. We also use Variable Bit Rate (VBR) UDP traffic, in which case *M* is distributed according to a log-normal distribution with geometric standard deviation $\sigma_M = 5$. In addition, we use a version of TCP Reno (implemented in our network simulator) to generate TCP traffic. Such traffic also consists of a succession of bursts generated by the source[9]. Since bursts transmitted in the Internet have been shown to be long-tailed [9], we use burst sizes distributed according to a log-normal distribution. As for UDP traffic, we use two sets of parameters: $\left(\overline{M},\ \sigma_M\right) = \left(2^{12}KBytes,\ 8\right)$ (where $\overline{M}$ represents the geometric average burst size, and $\sigma_M$ the geometric standard deviation for the log-normal distribution), used to model bursty (e.g. *http, ftp*) TCP traffic; $\left(\overline{M},\ \sigma_M\right) = \left(2^{11}KBytes,\ 5\right)$ for less bursty (e.g. *smtp, nntp*) TCP traffic.

We assume that the minimum available bandwidth offered to this queue is known, through the use of a Weighted Fair Queueing scheduling algorithm[10]. Since RQRD is designed to achieve both CSFQ's and RED's advantages, its performance is compared to both. Also, we compare RQRD to DT, since it is important to make sure that the improvement achieved warrants the complexity introduced. The simulation experiments conducted are grouped in three categories. In a future differentiated services network, UDP and TCP traffic will be separated in different queues; hence, we first assess the benefits of RQRD when applied to UDP traffic alone (Section IV-A), and TCP traffic alone (Section IV-B). Also, since RQRD is primarily intended to provide an adequate support of UDP and TCP traffic in today's networks, we include a set of experiments that demonstrate the performance of RQRD when applied to a mixture of UDP and TCP traffic (section IV-C).

### A. RQRD Applied to UDP Traffic Alone

We start with an experiment used in [12] where *N* sources, $S_0$ to $S_{N-1}$ transmit UDP flows with Source $S_i$ transmitting at an average rate $r_{in}^i = (i+1)\frac{C}{N}$. The goal is not to reproduce a realistic situation, but to have a common benchmark representing an extreme situation, with which the fairness achieved by different queue management schemes can be compared. We denote by $f$ the measure of fairness, defined by $f = \frac{\sum_{i=0}^{N-1}(r_{out}^i - \frac{C}{N})^2}{\sum_{i=0}^{N-1}[(2\frac{i+1}{N+1}-1)\frac{C}{N}]^2}$, where $r_{out}^i$ represents the output rate of flow *i*. In case the queue management scheme achieves perfect fairness, an average service rate of $\frac{C}{N}$ is allocated to each of the flows, leading to $f = 0$; conversely, a considerably unfair algorithm will allocate to each flow a portion of the link bandwidth that is proportional to its input rate yielding $f = 1$. As shown in Figure 3a, CSFQ is the most capable in providing fairness in such a setting. Clearly,

[9] A TCP connection is opened to transmit each burst; a source starts the transmission of a burst only after the correct receipt of the previous burst at the destination.

[10] In our simulations, we experiment with a single-queue system. (Accordingly, the minimum available bandwidth consists of the total link bandwidth.)

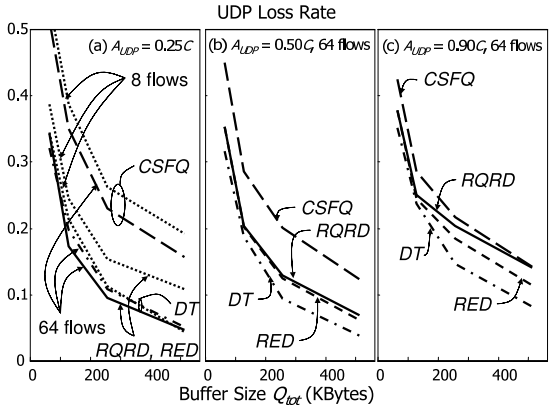


Fig. 4. Per-flow loss rate for $N$ VBR UDP flows versus $Q_{tot}$.

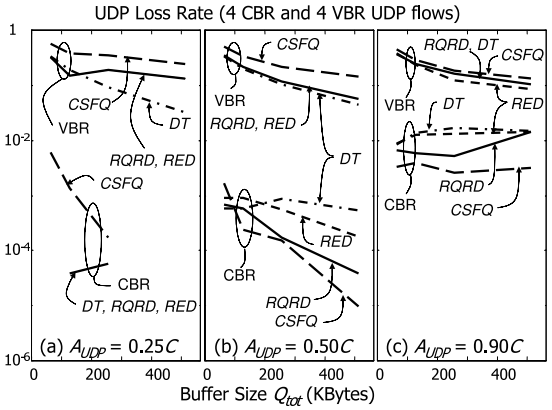


Fig. 5. Per-flow loss rate for $\frac{N}{2}$ VBR UDP flows and $\frac{N}{2}$ CBR UDP flows sharing a link. The rate proportion is $\frac{1}{3}$ CBR, $\frac{2}{3}$ VBR.

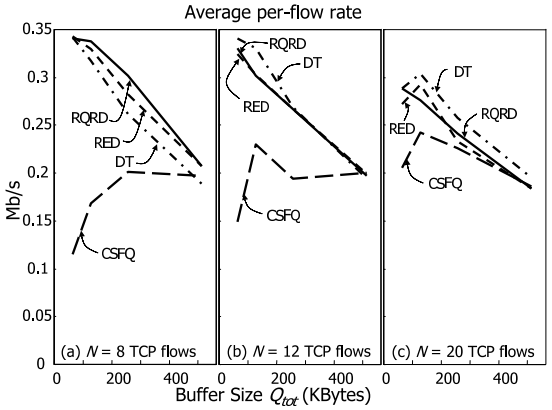


Fig. 6. Average per-flow rate for $N$ TCP flows sharing a link versus $Q_{tot}$.

both DT and RED are inadequate to provide fairness to UDP flows. In the case of RQRD, as attested by the results obtained with the *RED* and *ON-OFF* pairs, a discontinuity in either function $p_{low}(.)$ or $p_{high}(.)$ can result in RQRD becoming unstable. Using the *Ellipse* $(p_{low}, p_{high})$ pair instead (in which case both $p_{low}(.)$ and $p_{high}(.)$ are continuous), RQRD achieves a fairness that is comparable to that achieved by CSFQ as long as $Q_{tot}$ is larger than 128KBytes. Experimenting with different values of $Q_{min}$ and $Q_{max}$, the results show that the pair $(Q_{min} = \frac{1}{4}Q_{tot}, Q_{max} = \frac{3}{4}Q_{tot})$ bears the most homogeneous performance. Finally, $K$ should be set to a small value (e.g., 100ms) for RQRD to function correctly. (As seen in Figure 3b, RQRD performance degrades with increasing values of $K$.) In the remainder of these experiments (unless stated otherwise), we use $K = 100$ms for RQRD. (However, we use $K = 16\frac{Q_{tot}}{C}$ for CSFQ so extra burstiness is allowed when the buffer size is increased [12].) We conclude from this first experiment that unlike DT and RED, RQRD is capable of providing a theoretical fairness comparable to that obtained with CSFQ.

We now consider a realistic scenario in which $N$ statistically identical VBR streams share the link. In real life, this scenario represents, for example, the transmission of $N$ VBR video streams over the link. The important measure in this case consists of the loss rate experienced by video traffic. (See Figure 4.) We consider values of $A_{UDP}$ ranging from $0.25C$ to $0.90C$, where $A_{UDP}$ represents the average aggregate UDP arrival rate. The results show that in all cases, the best queue management scheme is simply DT. In fact, since traffic is bursty and sources are non responsive, preventive drop becomes harmful. We can see from the results that RQRD matches very closely RED's behavior and drops slightly more packets than DT. (This indicates that the congestion bit $c_Q$ is rarely set to 1.) Comparatively, with CSFQ, transient burstiness that could have been stored in the buffer is unnecessarily dropped instead, resulting in heavy packet drop, particularly when the average arrival rate is small[11].

In another experiment, we consider the mixture of CBR and VBR UDP traffic. This experiment models, for example, a situation where CBR and VBR video streams share the link. In our setting, one third of the traffic volume (in Mb/s) consists of $\frac{N}{2}$ CBR flows, while the remaining two thirds consists of $\frac{N}{2}$ VBR flows. We consider $A_{UDP}$ values ranging from 25 to 90% of $C$, and values of $N$ ranging from 8 to 64. As can be seen from Figure 5, CSFQ consistently drops VBR traffic the most. When the total arrival rate to the link is large, this behavior results in a lower drop rate for CBR traffic, a desirable result in line with the fairness objective that CSFQ is striving to achieve. However, when $A_{UDP}$ is small, then the excessive drop of VBR packets does not help reducing the drop rate of CBR traffic, which is the highest with CSFQ. Conversely, RQRD drops significantly less VBR packets. Also, when $A_{UDP}$ is large, RQRD is almost as capable in protecting CBR traffic as CSFQ. In fact, RQRD's ability to tailor its drop probability function based on the queue size using a hysteresis function allows it to differentiate more effectively between situations where congestion is transient (in which case packet drop is kept low) and those in which congestion is persistent (in which case adequate packet drop is applied).

### *B. RQRD Applied to TCP Traffic Alone*

We now investigate RQRD's ability to provide an adequate handling of TCP traffic. We first consider the simple case of $N$ statistically identical TCP flows mixed on a link. As can be seen from Figure 6, CSFQ performs poorly when $N$ is small: in fact, CSFQ falsely assumes that dropping packets at a rate $p$

[11] These results suggest that mechanisms aimed at taking into account queue size in CSFQ (that is, reducing $\alpha(t)$ with buffer overflow, and avoiding packet drop when the queue size is less than half the total buffer size, as described in Section II-B) are not sufficient. In order to confirm this fact, we repeat all experiments described in this paper with a version of CSFQ that is stripped from these two amendments: the results obtained with the two RQRD configurations match almost exactly.

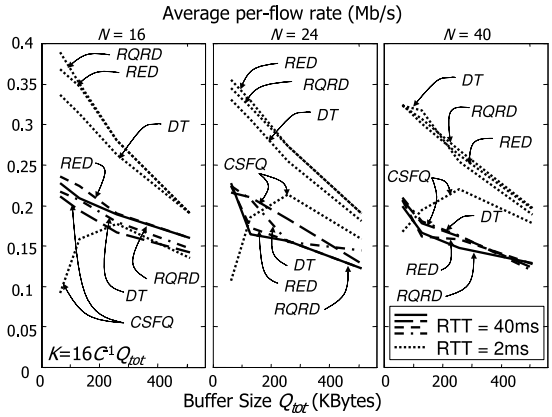


Fig. 7. Average rates for $N$ TCP connections sharing the same link; $\frac{N}{2}$ connections have a round trip time of 2ms, while the other $\frac{N}{2}$ have a round trip time of 40ms. ($K = 16\frac{Q_{tot}}{C}$.)

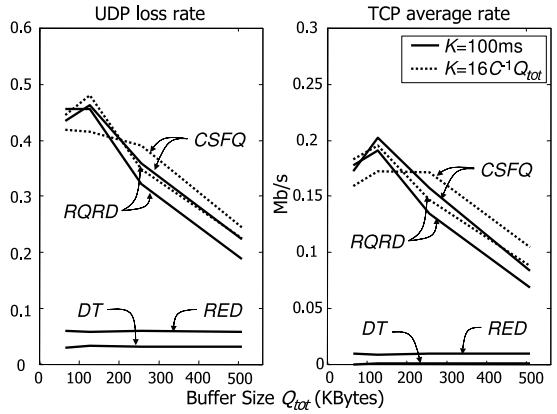


Fig. 8. UDP loss rate and TCP average rate per flow for one UDP CBR stream transmitting at $r_{in} = C$ mixed with 20 TCP flows.

from a flow with an incoming rate $r_{in}$ leads to an output flow with a rate $r_{out} = (1-p)r_{in}$. This is only true in the case of UDP traffic; however, in the case of TCP traffic, whenever $r_{in}$ exceeds $\alpha$, $p$ becomes positive irrespectively of the queue size, triggering an abrupt reduction in the TCP window size. The resulting sawtooth behavior leads to an average output rate $r_{out} \ll (1-p)r_{in}$. The performance gap between CSFQ and the other schemes gets smaller when either $N$ or $Q_{tot}$ increases. In fact, as $N$ increases global synchronization among TCP flows is reduced. Also, as $Q_{tot}$ increases, the reduction in throughput caused by the increase in round-trip time over-shadows the difference in throughput that results from the choice of a specific scheduling scheme.

Next, we consider the scenario where the $N$ TCP streams in question are divided into two groups (of $\frac{N}{2}$ streams each) which traverse links that have different propagation delays: $\tau = 1$ms and $\tau = 20$ms for Groups 1 and 2, respectively; as a result, TCP traffic from Group 1 is more aggressive, capturing most of the buffer resources. The experiment is intended to verify whether CSFQ is capable of insuring fairness among competing TCP flows. In fact, CSFQ falls short in this respect. For example, even though the plots corresponding to $N = 24$ in Figure 7 reveal that for $Q_{tot} = 256$KBytes, the use of CSFQ leads to an increase in the average rate of the flows belonging to Group 2 from 0.16 to 0.18Mb/s, at the expense of a reduction in the average rate of flows pertaining to Group 1 from 0.25Mb/s to 0.22Mb/s, these results are not consistent for all values of $N$ (e.g., $N = 16$ and $N = 40$). Also, as observed in the previous experiment, CSFQ performs poorly when either the number of flows is small or the buffer size is less than 256KBytes. Conversely, RQRD and RED are very close in their performance. Compared to DT, aggregate throughput is slightly improved by getting rid of global synchronization. Compared to CSFQ, aggregate throughput is in all cases higher, in most cases by a significant amount[12].

Similar results are obtained for an experiment in which two groups (of $\frac{N}{2}$ flows each) traverse links that have identical propagation delays ($\tau = 1ms$), but differ in their per-flow burst statistics ($\overline{M} = 2^{12}$ versus $2^{11}$ Bytes, $\sigma_M = 8$ versus 5).

*Summary.* Even though the use of CSFQ leads, in some situations to a desirable increase in fairness among TCP flows, this result is not consistent, and depends on factors such as the size of the buffer, the number of flows sharing the link and the setting of the parameter $K$ used in the algorithm. Perhaps more importantly, the results show that if either the buffer is too small, or the number of TCP flows is too small, CSFQ leads to a drastic decrease in aggregate throughput. In this case, the use of RQRD does not lead to an improved fairness among competing TCP flows, but provides an aggregate throughput that matches very closely that provided by RED, and which is never noticeably lower than that provided by Drop Tail.

### C. RQRD Applied to both UDP and TCP Traffic

In this section, we investigate RQRD's ability to deal with TCP and UDP traffic when mixed in the same FIFO queue. We start with a scenario in which we investigate RQRD's ability to protect TCP flows from a high volume, non-adaptive UDP flow. In this respect, we consider the following setting: one UDP flow, transmitting at a rate $r_{in} = C = 10$Mb/s is combined with $N$ TCP flows. As shown in Figure 8, with DT and RED, TCP traffic is starved almost totally by the ill-behaved UDP flow. On the other hand, both RQRD and CSFQ drop enough UDP packets so TCP throughput increases to reasonable levels[13]. (CSFQ performs slightly better than RQRD, owing to its ability to drop the ill-behaved UDP traffic consistently, proportionally to its input rate.) Similar results are obtained for $10 \leq N \leq 40$.

We now consider a set of realistic scenarios in which 16 UDP CBR streams, having an aggregate average arrival rate of 5Mb/s (that is, $\frac{1}{2}C$) share the link with a number of TCP flows $N_{TCP}$ ranging from 5 and 20. TCP traffic is heavily penalized in case CSFQ is used (see Figure 9); the gap between CSFQ and other queue management schemes is excessive when $N_{TCP}$ is small. Moreover, CSFQ drops significantly more UDP packets in this case (around 1%) than other schemes (0.5% for DT, 0.2% for

[12] As mentioned in Section IV-A in the context of UDP traffic, the similarity between the behavior of RQRD and RED suggests that with RQRD, TCP adapts quickly enough so that $c_Q$ is seldom set to 1.

[13] In the case of CSFQ, with $K$ set to 100ms, throughput is higher for small $Q_{tot}$, but decreases sharply when $Q_{tot}$ increases (because CSFQ acts too aggressively, forbidding the buffer to be fully utilized). As observed before, with $K$ proportional to $Q_{tot}$ instead, CSFQ is capable of taking advantage of the additional buffer space to bear a slight improvement in throughput.

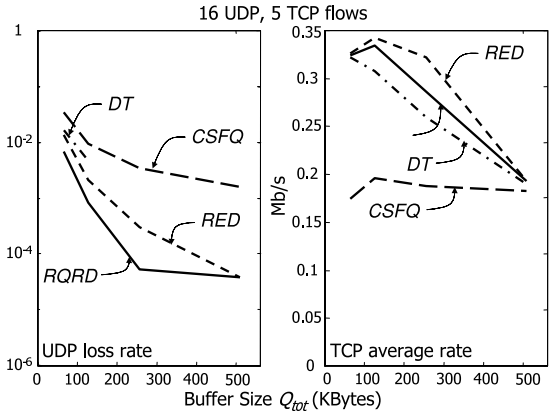


Fig. 9. UDP loss rate and TCP per-flow average rate versus $Q_{tot}$ for $N_{TCP}$ TCP flows sharing the link with 16 CBR UDP flows having an aggregate average rate of 5Mb/s. (16 UDP, 5 TCP flows.)

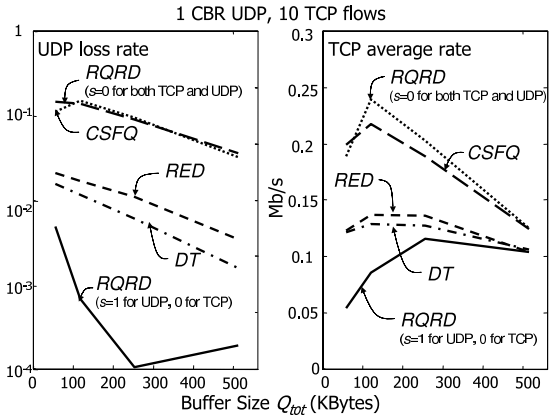


Fig. 10. UDP loss rate and TCP average per-flow rate in case one CBR UDP flow is mixed with $N$ TCP flows for different queue management schemes. (1 CBR flow mixed with 10 TCP flows.)

RED, and 0.08% for RQRD). In contrast, TCP per-flow average rates are significantly higher when RQRD is used instead. Also, RQRD drops significantly less UDP packets as long as $Q_{tot}$ is kept larger than 128KBytes (consistent with the results obtained in Section IV-A).

Finally, we illustrate RQRD's capability in differentiating between different traffic flows using the drop priority field. In that respect, we experiment with a scenario in which one 8Mb/s UDP flow shares the link with $N$ TCP flows. We consider two different settings for RQRD: in the first setting, we assign the drop precedence $s = 0$ for all flows; in the second setting, UDP traffic is assigned $s = 1$, whereas the $N$ TCP flows are assigned $s = 0$. As shown in Figure 10, while the aggregate throughput remains the same, TCP throughput is reduced in such a way that the UDP loss rate becomes insignificant.

*Summary.* In this section, we have demonstrated the key features that enable RQRD to be appropriate when applied to a FIFO queue that is shared by both UDP and TCP traffic. First, we show that unlike RED and DT, RQRD is capable of giving the same level of protection as CSFQ to TCP flows from unfriendly UDP flows, without incurring a loss of aggregate TCP throughput in situations where packet dropping is unnecessary. Also, we demonstrate the effectiveness of RQRD in shielding loss-sensitive UDP traffic from other traffic in the network.

## V. Conclusion

In this paper, we have presented a new buffer management scheme, *Rate and Queue Size controlled Random Drop* (RQRD). RQRD is based on the Core-Stateless Fair Queueing distributed architecture, but provides two drop precedences, and uses both rate information and queue size information to control the drop probability of packets. As a result, RQRD performs well in the presence of both UDP and TCP traffic. Also, it achieves a good compromise between providing fairness among different flows competing for a given link and maximizing the aggregate throughput on the link. Finally, it proves to be robust to both changes in internal parameter values and to a wide range of external conditions. These features render RQRD ideal in today's switches, where UDP and TCP traffic share the same buffer resources. In future routers, where UDP and TCP traffic will likely be provided with separate queues, the advantages of RQRD in a purely TCP context are the same as those provided by RED, and may not warrant the complexity introduced. However, RQRD's advantages are significant in a UDP context.

## References


[1] Y. Bernet, J. Binder, S. Blake, M. Carlson, B. Carpenter, S. Keshav, E. Davies, B. Ohman, D. Verma, Z. Wang, and W. Weiss, "A Framework for Differentiated Services," IETF working draft <draft-ietf-diffserv-framework-02.txt>, Feb. 1999.

[2] D. Clark and W. Fang, "Explicit Allocation of Best-Effort Packet Delivery Service," *IEEE/ACM Transactions on Networking*, vol. 6, no. 4, Aug. 1998.

[3] W. Feng, D. Kandlur, D. Saha, and K. Shin, "BLUE: A New Class of Active Queue Management Algorithms," Report CSE-TR-387-99, Computer Science and Engineering Division, University of Michigan, 1999.

[4] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397-413, Aug. 1993.

[5] E. Hashem, "Analysis of Random Drop for Gateway Congestion Control," Report LCS TR-465, Laboratory of Computer Science, M.I.T., 1989, p. 103.

[6] J. Ibanez and K. Nichols, "Preliminary Simulation Evaluation of an Assured Service," IETF working draft <draft-ibanez-diffserv-assured-eval-00.txt>, Aug. 1998.

[7] M. May, J. Bolot, C. Diot, and B. Lyles, "Reasons Not to Deploy RED," *Proceedings of IWQoS'99*, London, Mar. 1999.

[8] R. Pan, B. Prabhakar, and K. Psounis, "CHOKe, A stateless active queue management scheme for approximating fair bandwidth allocation", in *Proceedings of INFOCOM 2000*, March 2000.

[9] V. Paxson, "Empirically-Derived Analytic Models of Wide-Area TCP Connections," *IEEE/ACM Transactions on Networking*, vol. 2, no.4, pp. 316-336, Aug. 1994.

[10] N. Seddigh, B. Nandy, and P. Pieda, "Bandwidth Assurance Issues for TCP flows in a Differentiated Services Network," *Proceedings of GLOBECOM'99*, Rio de Janeiro, Dec. 1999.

[11] N. Seddigh, B. Nandy, and P. Pieda, "Study of TCP and UDP Integration for the AF PHB," IETF working draft <draft-nsbnpp-diffserv-tcpudpaf-01.txt>, Aug. 1999.

[12] I. Stoica, S. Shenker, and H. Zhang, "Core-Stateless Fair Queuing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks," *Proceedings of ACM SIGCOMM'98*, Sept. 1998.

[13] I. Stoica and H. Zhang, "Providing Guaranteed Services Without Per Flow Management," *Proceedings of ACM SIGCOMM'99*, Sept. 1999.

[14] C. Villamizar and C. Song, "High Performance TCP in ANSNET," Computer Communication Review, Vol. 24, No. 5, pp. 45-60, Oct. 1994.