Refined Design of Random Early Detection Gateways

Haining Wang and Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109-2122
E-mail: {hwx, kgshin}@eecs.umich.edu

Abstract—Random Early Detection (RED) was proposed as an active gateway queue-management mechanism. This paper proposes to alter the RED design guideline that unconditionally allows transient congestion, and evaluates its impacts. This unconditional allowance of transient congestion is shown to be harmful when the queue is near full, because it causes buffer overflow at a gateway. Buffer overflow at a gateway leads to the global synchronization and oscillation of traffic load on the network. To effectively prevent buffer overflow at a gateway, the RED framework is refined in such a way that the gateway can detect a transient congestion in a timely manner and take actions to quench it when the queue is near full.

Based on our simulation results, two enhancements are made in estimating average queue size. Equipped with these enhancements, the refined RED can strike a good balance between the allowance of transient bursty traffic and the avoidance of buffer overflow. Using extensive simulations, the refined RED is comparatively evaluated against and is shown to be superior to the original RED.

I. INTRODUCTION

The explosive growth of the Internet [14] makes it essential to devise and deploy effective congestion control at the transport layer. Since most of the Internet traffic is transported by the Transmission Control Protocol (TCP) [11], [13], TCP congestion control [7] plays a key role in avoiding the congestion collapse of the Internet. The current Internet architecture is featured by the end-to-end TCP congestion control, in which congestion control is accomplished solely by end-hosts. However, the gateway is recognized to be the most appropriate agent to detect incipient congestion and ensure fair allocation of network bandwidth [1]. The performance of end-to-end congestion control is expected to be greatly improved with the deployment of advanced gateway congestion-control mechanisms. The FIFO queueing with the drop-tail policy is widely employed in the current Internet gateways, scheduling packets in a First-In-First-Out (FIFO) manner and discarding those packets arriving when the gateway buffer is full. The drop-tail gateway is simple, scalable, and easy to implement. However, it is shown to distribute packet losses arbitrarily among TCP connections and also tends to penalize bursty connections [4]. The effects of global synchronization are reported to have been found in both one-way and two-way TCP traffic [12], [15], and thus lower aggregate throughput.

To address the drop-tail gateway problems, an Early Random Drop Gateway was proposed [6]. The Early Random Drop gateway drops arriving packets with a fixed drop probability if the queue length exceeds a certain drop level. Based on the Early Random Drop algorithm, the Random Early Detection (RED) algorithm was proposed by Floyd and Jacobson [5]. A RED gateway detects incipient congestion based on the computation of the average queue size, and randomly drops or marks arriving packets before the gateway buffer gets full. It keeps the average queue size low, while allowing fluctuations in the actual queue size in order to accommodate bursty traffic and transient congestion. To avoid a bias against bursty traffic and the global synchronization that exists in drop-tail gateways, the RED gateway uses randomization to choose which arriving packets to drop. The probability of dropping a packet from a particular connection is roughly proportional to that connection's share of the bandwidth through the gateway.

Lin and Morris [8] observed that RED allows unfair bandwidth sharing when a mixture of different traffic shares a link. Their key observation is that dropping packets from flows in proportion to their allocated bandwidth does not ensure the fairness in sharing bandwidth. The unfairness comes from the fact that at any given time RED enforces the same drop rate upon all flows regardless of their bandwidth shares. A modified version of RED, called Flow Random Early Drop (FRED), was proposed to handle this unfairness problem. FRED introduces "per-active-flow accounting" to enforce a drop rate on each flow that is dependent on the flow's buffer occupancy.

There are two functions of a congestion-avoidance mechanism at a RED gateway: one is to detect incipient congestion, and the other is to decide which connections to be notified of the congestion. FRED [8] focused on the second function, without considering the first. They improved the fairness of RED by modifying the way of notifying congestion at the gateway, but did not address the problem of detecting incipient congestion. The framework of FRED is the same as that of RED, and FRED uses the same parameters configuration as RED, thus leaving room for improvements.

This paper focuses on how to detect incipient congestion. We modify the RED design guideline of unconditionally allowing transient congestion (i.e., no negative feedback for transient congestion). Our analysis of the dynamics of TCP traffic and our simulation results show that transient congestion is harmful when the queue is near full. The queue weight \( w_q \) is used to control the rate at which the estimated average queue size reacts to the changing traffic load at gateways. In the original RED, \( w_q \) is preset and remains unchanged after its deployment. By contrast, in the proposed approach we monitor the dynamics of the actual queue and dynamically change the value of \( w_q \) according to the change of actual queue size. By reconfiguring \( w_q \), we de-
detect transient congestion in a timely manner when the queue is near full and take actions to quench it, thus effectively avoiding buffer overflow.

In accordance with the reconfiguration of $w_q$, the framework of the RED algorithm is also refined. The congestion-avoidance phase is extended and divided into a number of sub-phases. In contrast to the original RED in which the maximum drop probability is fixed, the refined RED dynamically adjusts the value of $\text{max}_{p}$ depending on which sub-phase the current average queue length belongs to. Since $\text{max}_{p}$ directly impacts the aggressiveness of the early detection mechanism and average queue buildup indicates incipient congestion, we increase $\text{max}_{p}$ as the average queue length increases from a lower sub-phase to a higher sub-phase. We used simulations to evaluate the performance of the refined RED in comparison with the original RED.

The direct outgrowth of the simulation study is two enhancements to the estimation of average queue size. One of the key observations from the simulation study is that the "surplus" between actual queue size and average queue size indicates the aggressiveness of traffic influx to the gateway. The higher the surplus, the burster the incoming traffic is. A sudden increase in the surplus indicates that the incoming bursty traffic is beyond the capacity of the gateway and the buffer overflow is imminent. The first enhancement is to dynamically adjust $w_q$ with the change in the surplus between actual queue size and average queue size.

The second enhancement is to solve the remaining problem in the original RED where the departure of a packet does not cause any change to the average queue size. FRED proposed a modification to correct it, but unfortunately, in some cases, it makes the situation even worse.

Section 2 briefly outlines the RED algorithm and the design guidelines for RED gateways. Section 3 describes the problem with RED gateways which is caused by transient congestion. Section 4 presents two enhancements to the estimation of the average queue size based on the simulation study. Section 5 describes the refined RED algorithm and its simulation results. Finally, the paper concludes with Section 6.

II. RED ALGORITHM

RED is to be deployed at a gateway for congestion avoidance and randomly drops packets before the gateway buffer is completely exhausted. RED aims to maintain high throughput and low delay by controlling the average queue size, and avoid global synchronization and a bias against bursty traffic.

The design idea of RED is very simple: two preset thresholds are used to detect incipient congestion and control the average queue size. Fig. 1 illustrates the general idea of RED. According to the estimated average queue length, a gateway operates in one of three different working states. When the average queue length is less than the minimum threshold, the gateway is in the green state. All incoming packets are processed and forwarded properly, and no packet is dropped. When average queue length is between the minimum and maximum thresholds, the gateway is in the yellow state. Arriving packets are randomly dropped with a probability that is a function of the average queue length.

\[ \text{avg} \leftarrow (1 - w_q) \text{avg} + w_q \text{q} \]

where $w_q$ is a constant parameter preset by RED that determines the sensitivity of RED to the fluctuation of actual queue size, and $q$ is the actual queue size. As $w_q$ varies from $\text{min}_{th}$ to $\text{max}_{th}$, the packet-drop probability ranges linearly from 0 to the fixed $\text{max}_{p}$. The final packet-drop probability increases slowly as the number of received packets increases since the last marked/dropped packet.

III. TRANSIENT CONGESTION CAN BE HARMFUL

A very important feature of the RED algorithm is the use of a low-pass filter to estimate the average queue size, allowing occasional congestion. Therefore, the RED algorithm is made non-responsive to transient congestion. At a gateway, many flows converge in, and the traffic seen by the gateway is a mixture of various types. TCP traffic is the most dominant component in the Internet [14]. Basically, there are two kinds of TCP transmissions: (1) long-lived bulk-data transmission; (2) interactive-data transmission or short-lived bulk-data transmission.

If the transient congestion is caused by long-lived bulk-data transmission, even in the absence of non-responsive or malicious flows, due to the dynamics of TCP window size, the transient congestion is very easy to become persistent in the near future if the RED gateway accommodates the transient congestion without any negative feedback to traffic sources. In the following subsection, the dynamics of the TCP window size are analyzed to verify this.

If the transient congestion is caused by short-lived transmissions or interactive-data transmissions, it may not be harmful when the queue is near full. Because such a flow will terminate or will soon become idle, transient congestion will not sustain. If $w_q$ is set too high, the RED algorithm will react too quickly and respond inappropriately to bursty traffic, resulting in underutilization and a bias against bursty connections.

When the average queue length is greater than the maximum threshold, the gateway is in red state in which every arriving packet is discarded. The behaviors of RED in green and red states are the same as those of drop-tail. Yellow is the key state in RED where the congestion-avoidance mechanism is implemented.

The estimation of the average queue size and the calculation of drop probability are two key components of the RED algorithm. The success of RED depends on how to estimate the average queue size and set the drop probability. The filter used to compute the average queue size is an exponentially-weighted moving average (EWMA) given by:
A. Analysis of the Dynamics of TCP Window Size

According to the TCP congestion-control scheme initiated in [7], the TCP window size $cwnd$ is set to the minimum of the congestion window size $cwnd$ and the receiver advertised window size $rwnd$. Before the occurrence of packet loss, the dynamics of a sender's TCP window size could be in the Slow-Start or Congestion-Avoidance phase:

Case 1: The sender is in the Slow-Start phase and $cwnd$ is much smaller than $rwnd$. Since $cwnd$ will be doubled per round-trip time, $cwnd$ will grow exponentially in the subsequent RTTs.

Case 2: The sender is in the Congestion-Avoidance phase and $cwnd$ is smaller than $rwnd$. Since $cwnd$ will be increased by one per RTT, $cwnd$ will grow linearly in the subsequent RTTs.

Once $cwnd$ gets larger than $rwnd$, $cwnd$ is set to $rwnd$. If transient congestion is allowed and no packet is dropped, the sender will assume that there is no congestion at all. At least the same amount of data will be sent in the subsequent RTTs. In the worst case where the sender is in the Slow-Start phase, the amount of data transmitted grows exponentially in the subsequent RTTs.

The buffer overflow at a gateway becomes inevitable after the elapse of several RTTs. Once the buffer space is exhausted, the RED gateway will degrade to a drop-tail gateway.

The simulation results in the next subsection confirm our analysis.

B. Simulation Setup and Results

The simulation experiments are conducted on the network in Fig. 2 using the ns [9]. The links in Fig. 2 are labeled with their bandwidth and one-way propagation delay. A couple of TCP connections (from $S_1$ to $K_1$) share a common bottleneck of 1.5 Mbps. The packet size in our simulation is fixed, which is set to 1 Kbytes. Each RED gateway has 25 packet buffers. The parameters of RED are set as the original: $\minthr = 5$, $\maxthr = 15$, $w_q = 0.002$, $\maxp = 0.02$. The version of TCP used in the simulation is TCP Reno, because it is built on UNIX BSD4.3 that is widely used but poor in recovering from multiple packet losses.

For each graph that shows the traffic dynamics in this subsection, the x-axis represents the time in seconds and the y-axis is the packet number mod 90. Marked on the graph are a packet's arrival and departure from the RED gateway $R_1$. Each dropped packet by the RED gateway $R_1$ is marked by "x." If there are $n$ connections, packets numbered 1 to 90 on the y-axis belong to the first connection, packets numbered 101 to 190 on the y-axis belong to the second connection, and so on. In the graphs that show average queue size, the solid line represents the queue of packets waiting to be transmitted on the link from $R_1$ to $R_2$. The dotted line represents the average queue size calculated by the RED gateway $R_1$.

In the first simulation scenario, three TCP connections compete and all the TCP traffic is long-lived bulk-data transmission. The TCP window size is set to 25. The simulation results in Fig. 3 (a) and (b) show that under the original parameters configuration and the framework of RED algorithm, the Slow-Start or Congestion-Avoidance algorithm can easily transform transient congestion to persistent congestion and exhaust the buffer.

In the second simulation scenario, there are three short-lived TCP transmissions. One long-lived bulk-data transmission is used as the background traffic. The simulation result in Fig. 3 (c) and (d) show that the aggregate short-lived traffic can also drive the buffer to exhaustion when the queue is near full, thereby causing a burst of packet losses. Since TCP Reno is poor in recovering from multiple packet losses, timeouts occur and are followed by a global synchronization.

We have attempted to use TCP New-Reno and SACK, instead of Reno, to test the performance of the original RED. Although TCP New-Reno and SACK have better capability to recover from multiple packet losses without causing retransmission timeouts, they still suffer frequent buffer overflows due to the inherent weakness of the original RED. Also, global synchronization and traffic load oscillation still occur frequently due to the frequent buffer overflows. The reason for this is that TCP New-Reno and SACK reduce their congestion window size to half after detecting packet losses, and hence, the buffer overflow at the bottleneck router drives all active TCP (New-Reno or SACK) connections to reduce their transmission rate by half, which leads to global synchronization. The only difference is that the fluctuation of the traffic load is smaller in New-Reno and SACK than Reno since fewer timeouts occur in New-Reno and SACK. The simulation results of New-Reno and SACK are not shown here due to the space limitation.

In the presence of multiple connections, congestion control algorithms can be used to manage the buffer space efficiently. However, in bursty traffic environments, traditional congestion control algorithms may not be sufficient. In such cases, other mechanisms such as RED can be used to manage the buffer space more effectively.

IV. ENHANCEMENTS TO THE DESIGN OF RED

To detect the initial congestion stage early, we could enlarge $w_q$ to increase the responsiveness of RED to bursty traffic. However, a faster increase of $w_q$ could result in over-reaction to short-lived bursty traffic even if its burstiness is within the bound of the remaining free buffer space, biasing against short-lived bursty traffic.

One key observation of the simulation study is that the surplus of actual queue size over average queue size reflects the burstiness of the incoming traffic. Based on this surplus, the gateway can gain a useful hint about the incoming traffic. A large surplus means bursty incoming traffic. The continuous growth of the surplus indicates that the incoming bursty traffic is beyond
We first compare the actual queue size with the average queue parity between packet departures and arrivals, we need to use a filter that the drop probability is also reduced. When the next-round traffic arrives, fewer packets are dropped. When the next-round traffic arrives, even though there is enough buffer space in the queue, the very beginning of the incoming traffic will still suffer from random packet drops due to the relatively high average queue size.

The second enhancement is how to calculate the average queue size in case of both packet arrivals and departures. FRED’s calculation for a packet departure is the same as that for a packet arrival. Unfortunately, in some cases this modification makes the situation even worse. Since the same low filter \( w_q \) is used to estimate average queue size when a packet departs, it implies that if the actual queue size shrinks rapidly, the average queue size slowly decreases. Thus, it is possible that the actual queue size is small but the average queue size is large. When the next-round traffic arrives, even though there is enough buffer space in the queue, the very beginning of the incoming traffic will still suffer from random packet drops due to the relatively high average queue size.

To correct this misadjustment, we propose a simple solution. We first compare the actual queue size with the average queue size. If the actual queue size is no less than the average queue size, then a packet departure causes the same effect on the average queue size as FRED. However, if the actual queue size is already smaller than the average queue size due to the speed disparity between packet departures and arrivals, we need to use a larger \( w_q \) to calculate the average queue size in order to reflect the rapid dequeuing. With decrease of the average queue size, the drop probability is also reduced. When the next-round traffic arrives, fewer packets are dropped.

The disparity is caused by transient quiescence or large reduction of the incoming traffic.

V. REFINING THE RED ALGORITHM

The refinement of the RED framework includes three parts: (1) the dynamic adjustment on the queue weight \( w_q \); (2) the fine-grained setting of \( \text{max}_p \); and (3) the refined framework of RED algorithm.

A. Dynamic Adjustment of Queue Weight

In the original RED, the queue weight \( w_q \) that is used to control the rate at which the gateway reacts to the congestion in the network, is preset to a low-pass filter. As shown in the preceding sections, the original RED cannot react to highly bursty traffic fast enough to prevent buffer overflow. On the other hand, if \( w_q \) is set too high, the RED algorithm will react too quickly to short-lived bursty traffic, causing bandwidth under-utilization and biasing against bursty traffic.

Based on the first enhancement in Section IV, we propose a simple mechanism to dynamically adjust the value of \( w_q \) upon each packet arrival. In the refined RED, the dynamics of the actual queue size is monitored. We introduce a new threshold called the warning line, when measuring the actual queue size upon arrival of a packet. It divides the setting of \( w_q \) into two phases. If the actual queue size is below the warning line, \( w_q \) is set exactly the same as the original RED. However, as soon as the actual queue size is on or beyond the warning line, \( w_q \) is set as follows: the higher the ratio of surplus to buffer size, the

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning/value of the parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>old_q</td>
<td>0.02</td>
</tr>
<tr>
<td>buf_s</td>
<td>buffer size</td>
</tr>
<tr>
<td>warn_line</td>
<td>half of the buffer size</td>
</tr>
<tr>
<td>q</td>
<td>actual queue size</td>
</tr>
<tr>
<td>avg</td>
<td>average queue size</td>
</tr>
<tr>
<td>diff</td>
<td>surplus of actual queue size over average queue size</td>
</tr>
<tr>
<td>ratio</td>
<td>ratio of surplus over buffer size</td>
</tr>
<tr>
<td>R</td>
<td>the integer part of ((10^{R} \text{ratio}))</td>
</tr>
<tr>
<td>new_q</td>
<td>queue weight</td>
</tr>
</tbody>
</table>
larger of the queue weight:

\[ w_q = \begin{cases} 
    \text{old}_q, & R \in [0, 0.1) \\
    \text{old}_q \times 4, & R \in [0.1, 0.2) \\
    \text{old}_q \times 8, & R \in [0.2, 0.3) \\
    \text{old}_q \times 12, & R \in [0.3, 0.4) \\
    \text{old}_q \times 16, & R \in [0.4, 0.5) \\
    \text{old}_q \times 20, & R \in [0.5, 1] 
\end{cases} \]

where \( R \) is the ratio of surplus to buffer size. We recommend that the warning line be set to half of the buffer size.

Upon departure of a packet from the queue, \( w_q \) is set according to the second enhancement in Section IV. If the actual queue size is smaller than the average queue size, \( w_q \) is set to 0.2, instead of 0.02, in order to be responsive to rapid dequeueing. A detailed algorithm for computing \( w_q \) is shown in Fig. 4, and the meaning of the parameters is shown in Table I.

The key point here is to detect the initial stage of congestion quickly when the queue is near full and the incoming traffic is highly bursty, but still absorb short-lived or interactive traffic as best as it can.

B. Fine-Grained Setting of Maximum Drop Probability

The performance of RED is very sensitive to the maximum drop probability \( \text{max}_p \), which determines the aggressiveness of RED towards incoming packets when it is in yellow state. The motivation of varying \( \text{max}_p \) according to the change of average queue size is to lower queueing delay and avoid buffer overflow. Since the build-up of average queue size indicates the imminence of persistent congestion, the RED gateway should be more aggressive when the average queue size increases.

The fine-grained setting of \( \text{max}_p \) is closely related to the refinement of the RED framework. The refined framework of RED has salient features, including:

- Decreasing \( \text{min}_th \) and increasing \( \text{max}_th \) extends the yellow area. The green and red areas are reduced to 15% and 25% of the total buffer space, respectively, thus leaving up to 60% of the buffer space for the yellow area. This in turn enlarges the scope of congestion avoidance.
- The yellow area is evenly divided into several sub-phases in each of which the value of \( \text{max}_p \) is different. As the average queue size increases, \( \text{max}_p \) is increased discretely. The number of the sub-phases depends on the buffer space covered by the yellow area and the buffer space per sub-phase. We suggest that each sub-phase be assigned to 10% of the buffer space. Since 60% of the buffer space belongs to the yellow area, there are 6 sub-phases in the yellow area of the refined RED.

Note that \( \text{max}_p \) in the original RED remains fixed regardless of the estimated average queue size. The packet-drop probability \( P_d \) is set to \( \text{max}_p \times \text{avg} / (\text{max}_th - \text{min}_th) \). With the change of average queue size, \( P_d \) varies linearly from 0 to \( \text{max}_p \). However, the fixed maximum drop probability of the original RED does not work well for different traffic loads. In the refined RED, the setting of the maximum drop probability is fine-grained. With the change of average queue size, \( \text{max}_p \) is dynamically switched to different settings. Within each sub-phase, the value of \( \text{max}_p \) is also fixed and \( P_d \) varies linearly from 0 to \( \text{max}_p \). The dynamics of the maximum drop probabilities in the refined RED is shown in Fig. 6. Note the figures are not drawn to scale.

C. Discussion

Instead of continuous adjustments of \( w_q \) and \( \text{max}_p \), we discretely set \( w_q \) and \( \text{max}_p \) to different levels. At each level, \( w_q \) or \( \text{max}_p \) is fixed to absorb the disturbance caused by short bursty traffic. However, if the burstiness of the incoming traffic lasts long enough, the gateway should automatically push \( w_q \) and \( \text{max}_p \) to a more aggressive level in order to quench the burstiness of the incoming traffic for the avoidance of buffer overflow.

D. Simulation Results

We use the same simulation setup in Section 3.2, and ten TCP connections share the common bottleneck link. The first con-
Bursty Traffic

We now show that the refined RED does not have bias against bursty traffic. We model the bursty traffic by using TCP connections with smaller windows (10 vs. 30) and longer RTTs (100 ms vs. 26 ms). Among the ten TCP connections in our simulation, two of them are bursty flows. Fig. 9 shows the dynamics of a bursty TCP. Clearly, the refined RED preserves one of the key advantages of RED avoiding bias against bursty traffic. Furthermore, the bursty traffic in the refined RED also benefits from the avoidance of buffer overflows. It has less timeouts and higher throughput than the bursty traffic in the original RED.

VI. RELATED WORK

A self-configuring RED gateway was proposed in [3], which adaptively changes the maximum drop probability \( \text{with}_p \) depending on the variations in average queue size. However, only \( \text{with}_p \), which indicates the aggressiveness of RED, is adaptively changed; \( \text{with}_p \), which indicates the responsiveness of RED, is still kept constant. Also, the adaptive change of \( \text{with}_p \) is coarse-grained, switching \( \text{with}_p \) from normal to conservative or aggressive.

Stabilized RED (SRED) was proposed in [10], which statistically estimates the number of active flows and adjusts the drop probability based on this estimation. Unlike in RED, there is no computation of average queue size in SRED. The drop probability depends upon the actual queue size and upon the estimated number of active flows.

RED with In/Out bit (RIO) was described in [2]. By configuring two sets of RED parameters, one for In-packets and one for Out-packets, RIO discriminates against Out-packets in case of congestion. The In/Out bit is set selectively to control which packets are favored during congestion. However, the two different settings of RED parameters in RIO are still static, which do not adaptively adjust to the change of the observed traffic load.

---

3Note the buffer size at the RED gateways is set to 25, and \( \text{with}_{18} \) is enlarged to 18, instead of 15 in the refined RED.
Transmit congestion is shown to be harmful when the queue of the gateway is near full. To prevent the buffer overflow, a gateway must therefore be responsive to transient congestion when the incoming traffic is highly bursty and the free buffer space falls below the warning line. In such a case, unconditional accommodation of transient congestion causes frequent buffer overflows and traffic load & delay oscillations.

We proposed a simple and efficient method to measure the burstiness of incoming traffic. Based on this measurement, we dynamically adjust the estimation of the average queue size. The setting of the maximum drop probability is finer-grained than the original RED. As the average queue size changes, the maximum drop probability is dynamically set to different aggressiveness levels. Our simulation results have shown the superiority of the refined RED to the original RED. It effectively prevents buffer overflow, reduces the oscillation of the network delay and traffic load, yet accommodates bursty traffic as best as it can.

REFERENCES