Harsh RED: Improving RED for Limited Aggregate Traffic

Ilpo Järvinen, Yi Ding, Aki Nyrhinen and Markku Kojo
Department of Computer Science
University of Helsinki
Helsinki, Finland
Email: {firstname.lastname}@cs.helsinki.fi

***Abstract*—A bottleneck router typically resides close to the edge of the network where the aggregate traffic is often limited to a single or a few users only. With such limited aggregate traffic Random Early Detection (RED) on the bottleneck router is not able to properly respond to TCP slow start that causes rapid increase in load of the bottleneck. This results in falling back to tail-drop behavior or, at worst, triggers the RED maximum threshold cutter that drops all packets causing undesired break for all traffic that is passing through the bottleneck. We explain how TCP slow start, ACK clock and RED algorithm interact in such a situation, and propose Harsh RED (HRED) to properly address slow start in time. We perform a simulation study to compare HRED behavior to that of FIFO and RED with recommended parameters. We show that HRED avoids tail-drop and the maximum threshold cutter, has smaller queues, and provides less bursty drop distribution.**

***Keywords*-Harsh RED (HRED); Random Early Detection (RED); Limited aggregate traffic; TCP slow start; Non-core bottleneck; Constant bit-rate (CBR) traffic; Wireless broadband**

I. Introduction

Random Early Detection (RED) [1] is an algorithm for a router to perform Active Queue Management (AQM) [2] by reacting pro-actively to an incipient congestion. RED aims at keeping long timescale queuing at low level, while allowing short timescale burstiness to occur. To achieve this, RED allows a number of its parameters to be configured in order to take into account heterogeneous path characteristics present in the Internet. However, it is generally perceived as challenging task to come up with right parameterization for RED [3], [4], [5], [6], [7].

Typically RED is thought to be placed on a router that aggregates significant amount of traffic. Previous studies have mainly provided suggestions on how to improve RED when the bottleneck is on a core router instead of residing near the edge of the Internet. As such, the effect of a single TCP flow in slow start to the overall utilization of such bottleneck is relatively small [8], [9]. This, however, leaves one essential part missing from the earlier proposals; they all are too "gentle" to handle fast changes due to TCP slow start, when the aggregate traffic is limited.

In this paper, we focus on a common case where the bottleneck is close to the edge of the Internet and even a single flow alone is able to saturate the link in question. Such link is subject to highly varying load changing rapidly from light load to overload in time as short as two round-trips because of the TCP slow start. This is very common behavior with mobile cellular and home access for example, as such bottleneck links typically carry traffic of a single or a few users only and TCP flows tend to be short, staying in slow start mainly. RED with recommended set of parameters [1], [10] is too slow to respond to such a limited aggregate traffic due to the small weight (w_q) used in calculating the moving average for the RED queue. Therefore, it either falls back to tail-drop when the physical queue is set to a small value, or if the physical quuee size is big, the RED queue average exceeds the RED maximum threshold (th_{max}) causing large number of drops.

According to our knowledge, there are few studies on RED with limited aggregate traffic, especially with various buffer size settings. We find out that something based RED algorithm itself is not satisfactory due to the nature of RED to over-correct slow start when the number of flows it regulates is small. But at the same time it is an algorithm that exist in a large number of hardware that is already deployed. Therefore, we find it logical to try to take advantage of RED instead of coming up with a new queue controller. Even though such controller might better suited to the task, it would be unlikely to be deployed in the near (or far) future.

For the purpose of countering slow start, we propose Harsh RED (HRED). By taking advantage of well-defined TCP slow-start behavior, we make HRED queue average to cross HRED minimum threshold (th_{min}) on a timely manner in order to begin "a count-down" for actual HRED dropping. It is made possible by setting the HRED weight (w_q) and the upper bound for the marking probability (max_p) to a very large value compared with the recommended values [1], [10]. In addition, we ensure that HRED drops before the physical buffer can run out during the final round-trip of the slow start.

We conduct a simulation study in an environment mimicking a wireless broadband access with a typical workload of Web browsing. In Web pages individual objects are typically small and the flows transferring them spend most or all of their lifetime in slow start which motivates our focus. Two wireless link types resembling Enhanced Data rates for GSM Evolution (EDGE) [11] and High-Speed Packet Access (HSPA) [12] were chosen. Besides avoiding tail-drop fallback and the maximum threshold cutter that occurs with RED,

HRED successfully reduces the queue length and provides less bursty drop distribution than FIFO or RED with the recommended parameters.

The rest of the paper is organized as follows. First we introduce related work in Section II. Then we focus on TCP slow start and its interactions with RED algorithm in Section III and derive Harsh RED from that knowledge in Section IV. We present simulation result where we compare Harsh RED performance against FIFO and recommended set of RED parameters in Section V. Finally, we discuss aspects related to Harsh RED and RED algorithm in Section VI and conclude the paper in Section VII.

II. Related Work

The initial guidelines to configure RED parameters are provided by the designers of RED [1], [10]. As pointed out in [3] the standard setting of RED exhibits no significant advantage compared to tail-drop and in overloaded situation even suffers both high delay and high packet loss. To seek a solution, several studies investigate the RED parametrization from different perspectives with the support of queuing theory and simulation tests [13], [14], [15]. A modification to reduce the RED queue average faster when the instantaneous queue shrinks is presented in [13], but it does not discern whether the queue average is trending upwards or downwards. Therefore, it mis-operates during slow start as empty or decreasing queue does not guarantee that the system load is decreasing because of RTT long feedback loop in the TCP response.

Various RED alternatives have been proposed to leverage the limitation of RED in challenging environments, including Adaptive RED (ARED) [16], [17], Gentle RED (GRED) [18] and Stabilized RED (SRED) [19]. In ARED, the upper bound on the marking probability (max_p) is adjusted to target a value for queue average within the RED minimum and maximum threshold range over long timescales. However, in short timescales this adaptation adds yet another component that is behind the current situation. As TCP slow start is very rapid, ARED adaption cannot address it. In addition, the authors of ARED [17] acknowledged RTT effect but incorrectly set weight (w_q) to a small value based on ten RTTs of 100 ms each which typically corresponds to multiple RTTs (or even to eternity in environments such as data-centers). As such, ARED queue average responds too slowly similar to standard RED. GRED [18] is identical with the standard RED below the maximum threshold which is where RED operates during the initial slow-start from idle. In addition, ARED is shown to yield poor performance with Web traffic [7]. SRED [19] focuses on finding out the number of flows which is significant for TCP congestion avoidance mode but quite irrelevant with slow start because the aggregated window growth rate in slow start does not depend on number of flows. In addition, Data-Center TCP (DCTCP) [20] is a TCP variant that includes end-host modifications and uses RED implemented on routers in a degenerated fashion. Because of the need for end-host modifications, we find DCTCP unsuitable for other than isolated environments such as data-centers as otherwise fairness issues would arise.

To evaluate RED performance, research at INRIA [21], [6] provides an analytical model and experimental results for tuning RED by taking into account both queuing delay and drop rates at the router. Based on their analysis, RED routers with standard parameters behave similar to tail-drop routers when queue length is close to the maximum threshold. Other experiments focusing on Web traffic indicate that for loads below 90% link utilization, RED exhibits little improvement in response time compared to tail-drop [5].

III. Anatomy of RED

RED maintains an exponentially weighted moving average (EWMA) of queue size. Typically, RED queue average (avg) is calculated when a packet arrives:

$avg \leftarrow (1 - w_q)avg + w_q q_{len}$

The EWMA weight is specified with w_q. A large weight makes the average to follow instantaneous queue length q_{len} more closely, while a small weight makes the queue average response slower. Purpose of the average queue size is to allow some burstiness, yet keeping the queue small even if the burst turns out to be non-temporary in nature.

A TCP flow has two modes of operation called slow start and congestion avoidance both affecting RED queue average differently. In the slow start TCP probes the network with an exponentially increasing window in order to reach an equilibrium [8]. Congestion avoidance is much "friendlier" to the network imposing only additive increase per RTT at most. TCP flows transferring Web objects spend most of the transfer in slow start as Web objects are typically short. Therefore we mainly focus to slow start in the analysis of this paper. If a TCP flow does not use Delayed ACKs [22], the congestion window of the TCP flow is doubled on each RTT (with Delayed ACKs, factor of 1.5 increase is expected). It implies that the load on the bottleneck can grow from 50% to 100% over a single RTT and go much beyond on the next RTT. Therefore, RED cannot wait many RTTs until marking or otherwise the load and queue would be very large. On the other hand, marking too early leads easily to under-utilization. Even if more than one flow is sharing the link, the growth rate is the same because every packet transmitted over the bottleneck triggers at most a single ACK clock tick regardless of flow.

At the main operating region above th_{min}, RED marking probability is composed of two components. First, the initial probability $p_{initial}$ is a linearly increasing function of average queue size between th_{min} and th_{max}. Second, final marking probability is formed using uniform random variables:

$p_{final} = \frac{p_{initial}}{1 - count * p_{initial}}$.

In this function the number of packets ($count$) that have arrived since the last marking or since the avg exceeded th_{min} is quite significant factor. It rapidly increases the final marking probability when $count$ approaches $1/p_{initial}$. The constant max_p can be used for the initial marking probability to adjust the weight of these two components. RED has two other operating regions. First, when the queue average is

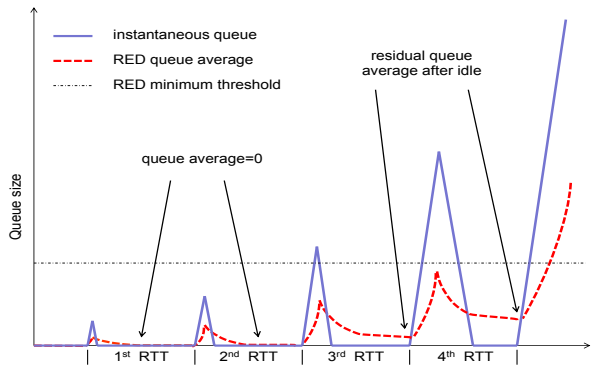

Fig. 1: Instantaneous and RED average queue during slow start

below th_{min}, no marking is performed and the $count$ is reset. Second, if the queue average is above th_{max}, all packets are marked. This mode is "a safety valve" of RED allowing return to main operating region if the load is too extreme for the main operating region to handle. In general, RED should not be configured such that this mode is ever used under normal operation like TCP slow start.

Figure 1 shows with a solid line the instantaneous router queue when TCP is in slow start. The router queue grows even when the link is not even near of being fully utilized due to the sender injecting packets on each RTT during a "slow-start burst" at higher rate than the bottleneck link is able to drain the queue. Without Delayed ACKs, the surplus is equal to the rate of ACK clock (half of the double). Yet, the queuing is only transient in nature and on each RTT an idle periods follows the slow-start burst. The transient spikes have an increasing magnitude and the idle period becomes shorter because of exponentially increasing TCP congestion window. Eventually the link becomes saturated and instantaneous queue will not reach zero anymore.

Each slow-start burst increases the RED queue average shown in Figure 1 with the dashed line. When the buffer becomes empty, the average is calculated using approximation formula that simulates RED clock ticks using phantom packets between the time when the last packet left the queue ($time_{queueempty}$) and the next real packet arrival at now:

$avg \leftarrow (1 - w_q)^{(now - time_{queueempty})/s} avg$.

The time to transmit a phantom packet (s) is configurable. All phantom packets are assumed to be of the same size (some form of average packet size should be used in RED configuration). When a slow-start burst is small, the average falls back to zero before the burst on the next RTT begins. If idle period is short enough, this formula leaves significant residual queue average.

IV. Harsh RED

As a basis for Harsh RED (HRED), we take the assumption that there is a slow start which causes a rapid utilization increase from 50% to 100% utilization (assuming no delayed ACKs). Recommended RED parameters [10] are too slow to react allowing queue to build up. HRED should operate on region below th_{min} only when the link is not saturated and quickly enter the main operating region as soon as the link becomes overloaded. A transient slow-start burst should not, or should not be likely to, cause losses while the queue is expected to drain slightly later. At the same time, HRED must be "warmed up" during these early RTTs. This warm-up ensures that transition to the main operating region happens soon after the load changes from under-utilization to overload. It is not enough to let HRED to perceive this from the ever growing queue because of inherent sluggishness in reaching a high enough $count$ value to force a drop. Instead, HRED must enter the main operating region earlier in the "final burst", which will not let the queue to return back to zero. We take advantage of the residual queue average (see Figure 1) in order to enable rapid transition to the main operating region. Therefore, we take half of the bandwidth-delay product (BDP) of the link and select w_q such that dropping is unlikely to occur already with a slow-start burst that equals to the half of the link BDP in size. At the same time we ensure thata significant residual queue average remains after the idle period.

In order to really take advantage of the pro-active dropping instead of falling back to tail drop when the physical buffer becomes full, the router buffer is set to a large value such that it effectively never overflows (except due to extremely unresponsive traffic). th_{max} is also set such that it is never reached due to slow start alone. Reaching th_{max} ever should be highly discouraged because it tends to drain "the pipe" due to consecutive dropping, a behavior worse than FIFO as the queue average can reside above th_{max} even when the instantaneous queue is nearly or totally empty. In addition, the feedback latency must be taken into account as the slow start continues until the loss signal reaches the sender and is only then reflected as a lower sending rate towards the router. In total, the feedback loop from router to receiver to sender and back to router takes one RTT and a TCP sender in slow start doubles the sending rate during this RTT.

The traditional wisdom in configuring RED seems to be "being gentle" to avoid dropping much too early. The original RED authors state that "there should never be a reason to set max_p greater than 0.1" [1]. Also, there is even a RED variant [18] carrying this kind of "desired" behavior in its name. However, with slow start the "being gentle" approach is exactly the wrong one. Instead, RED should treat any slow-start probing near the link saturation "harshly" to prevent the congestion from escalating into higher dimensions. The fact is that slow-starting will not cease until drops are induced, and it is better to do that sooner than later. In addition, the dropping needs to be aggressive enough to successfully counter common-mode slow-start of multiple flows. If only a few drops are induced, only a few flows will end their slow start which still leaves the others to continue in slow start with the exponential increase. Dropping sooner means that queue average is not even close to th_{max} just yet because it naturally lags behind the instantaneous queue sizes. Therefore in practice, the initial dropping probability will not reach a

TABLE I: RED and HRED Paramenters

	EDGE		HSPA	
	RED	Harsh RED	RED	Harsh RED
th_{max}	9	40	20	50
th_{min}	3	3	5	3
w_q	0.002	0.2	0.002	0.02
max_p	0.1	0.65	0.1	0.65

value close to max_p.

In order to make HRED to behave "harshly", we need to pay attention to when at latest the HRED is guaranteed to drop as that defines the upper bounds for the queue growth (albeit multiple flows make the actual analysis more complicated). RED configuration guideline [1] only pays attention to the opposite question, when at earliest, the dropping can occur. Because the final dropping probability and th_{max} are interlocked, an increase in th_{max} will also delay the latest point of guaranteed drop although not linearly. At the same time, $count$ is important in determining when the rapid increase in final dropping probability is taking place. The growth of $count$ can be controlled by allowing the th_{min} to be crossed early, i.e., to have a small th_{min}. This leaves us only max_p as a free parameter which as a result should be made large.

During the feedback RTT the queue average keeps increasing that leaves the HRED into even more aggressive state for the RTT following the feedback RTT. Therefore, HRED induces drops on the subsequent RTT more frequently than it did during the initial reaction RTT. Such unnecessary drops cause extra window reductions which can lead to under-utilization. A smaller number of flows increases the likelihood of under-utilization because the impact of a single window reduction to the load is high. This over-reaction is an inherent flaw in EWMA queue average. We took it into account by selecting parameters that drop slightly later than would have been necessary if only a single window reduction is expected. The key HRED parameters are summarized in Table I.

V. Simulation Experiments

In order to evaluate the performance of Harsh RED, we conduct extensive simulations on a variety of workloads and link access settings using the network simulator ns-2 [23].

A. Simulation Setup

We decide to keep the simulation topology simple and yet representative roughly targeting at two cellular wireless access environments, EDGE and HSPA, inherently with limited aggregate traffic on the access link. We select a number of workloads to represent the scenarios, where the main traffic is Web transfers with multiple connections downstream from an Internet content server to the mobile host. Figure 2 presents our testing topology. The wireless access link between the mobile host and the last-hop router is the bottleneck link representing typical cellular access link characteristics. To investigate the queuing behavior and avoid loss noise, all the links are simulated as loss-free. While larger delays than 11 ms towards the content server can be easily encountered in the

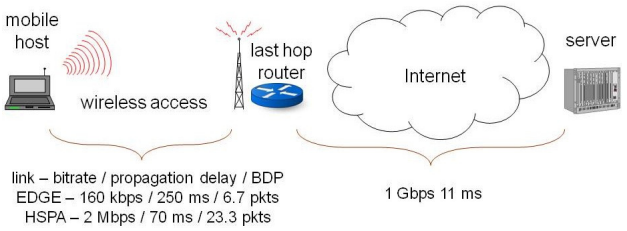

Fig. 2: Simulation Topology

Internet, the delays of the selected wireless links are expected to dominate RTT except in the case of inter-continent flows.

We compare the proposed HRED queue management against that of FIFO and RED. To compare the impact of different queue settings, we select RED and HRED with the parameters given in Table I with byte-based dropping probability calculation. Two different router buffer sizes are tested: 2BDP that is two times the bandwidth-delay product (BDP) of the link and Large buffer size. The Large buffer is based on the wireless link parameters with the goal to make it "large enough" for HRED queue to not overflow it. As one of the goals with HRED is to have such large buffer, the smaller 2BDP-sized buffer is not tested with it. HRED is also tested with Explicit Congestion Notification (ECN) [24].

TCP performance is tested with two workloads: 1) one background bulk TCP flow with a burst of later starting TCP flows and 2) a burst of n TCP flows with a burst of n later starting TCP flows. The background bulk TCP flow and the first n TCP flows are started at zero seconds. The background bulk TCP flow is left to settle into congestion avoidance mode of operation before the later TCP burst is started. Each burst consist 1, 2, 6, or 18 TCP flows that start simultaneously. The total size of the TCP burst is 360 kB divided equally between the flows in the given burst. To measure delay effects, we include constant bit-rate (CBR) traffic with payload rate of 16 kbps for EDGE and 64 kbps for HSPA to compete with the TCP traffic. The CBR start time is distributed to occur 0-200 ms earlier than the later starting TCP burst and it lasts for 15 seconds with EDGE and 1.5 seconds with HSPA. The CBR flow is not ECN-capable. Each test case is repeated 100 times using random distribution for start time of the later TCP burst. Due to the limited space, the delay effects are only presented for the cases with 2 or 6 flows in the TCP burst.

Performance metrics for our measurements include the TCP burst elapsed time, oneway delay, and loss-pair distance. The TCP burst elapsed time for a TCP burst is defined as the time between sending the first TCP SYN and receiving the last ACK packet for the last to complete flow within the burst. The loss-pair distance indicates the distance in packets between two losses which is important for CBR audio traffic as codecs tend to be able to conceal only gaps with limited duration. Compared to network-centric metrics such as link utilization, the selected metrics represent better the factors affecting the end user experience. The burst elapsed time is presented using figures that show median and quartiles (25th

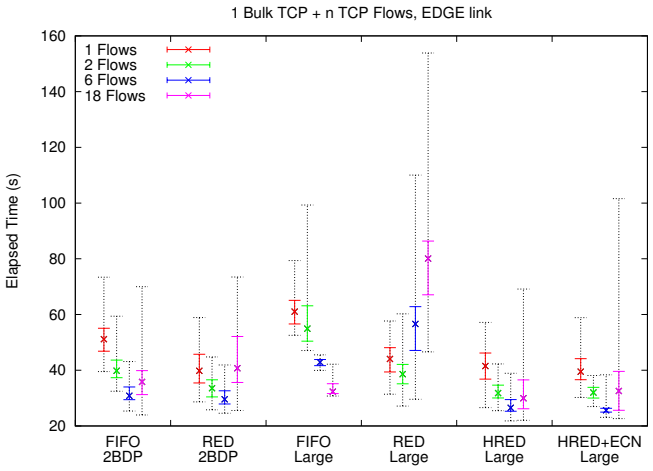

Fig. 3: Elapsed time for the burst of TCP flows over an EDGE link

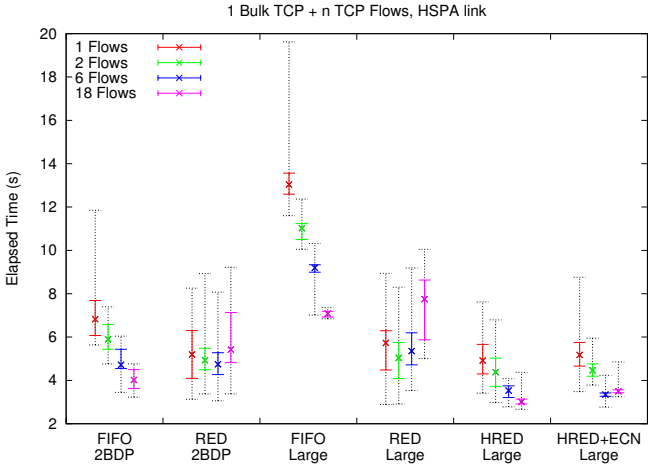

Fig. 4: Elapsed time for the burst of TCP flows over an HSPA link

and 75th percentiles) using solid errorbars, and the minimum and maximum using dotted errorbars.

B. TCP Performance Results

First we look at the case with a background Bulk TCP flow (1 Bulk TCP) and a burst of TCP flows. The elapsed time of the TCP burst over the EDGE link is presented in Figure 3. With RED+2BDP when the burst starts, the queue average is already on the RED operating region because of the background bulk TCP. Therefore, the queuing times are short with RED+2BDP and the TCP burst is able to complete faster than with FIFO, except with 18 flows. With 18 flows, RED+2BDP with too late pro-active drops causes exponential RTO backoffs more likely than with FIFO. However, as the BDP of the link is less than 7 packets, 18 concurrent flows is not sustainable even with TCP congestion windows of one segment and TCP retransmission timeouts (RTOs) are triggered with FIFO+2BDP too. With 1 or 2 flows in the TCP burst, RED+Large has shorter burst elapsed time than FIFO+2BDP similarly to RED+2BDP. However, when the number of flows is either 6 or 18, the elapsed time increases because a th_{max} cutter phenomenon prevents all transmission for multiple seconds. The th_{max} cutter occurs when the queue average increases above th_{max} as a result of long instantaneous queue. New packets cause further increase in the queue average. When the queue average is above th_{max}, RED considers the router overloaded and drops everything until the average returns below th_{max}.

HRED is better than FIFO for similar reasons as RED+2BDP but also manages 18 flows better because drops are better distributed among the flows causing less exponential RTO backoffs to occur and the completion time of the flow to finish last is improved significantly. HRED provides shorter burst elapsed times than RED+2BDP when there is more than one flow because it favors less the first flow in a burst, allowing the later flow in the same burst to complete sooner. The improvement in the median elapsed time is 5.3-26.4%. With one flow the significant part of the transfer happens in TCP congestion avoidance mode and the link becomes occasionally under-utilized which makes the HRED median elapsed time 4.3% longer than that of RED+2BDP.

With HRED+ECN the burst elapsed time tends to be shorter than with HRED. However, in the two flows case the median elapsed time is slightly longer while both quartiles show an improvement. In addition, with 18 flows HRED is better than HRED+ECN because HRED+ECN marks instead of dropping which allows the queue to grow until th_{max} cutter occurs. The situation resolves more quickly than with RED+Large because of the smaller gap between instantaneous queue length and th_{max} and the traffic is not fully stopped. In addition, some spurious RTOs [25] occur with HRED+ECN due to the rapid arrival of TCP initial window data for several flows and also some retransmission are dropped due to inability of using ECN with TCP retransmissions. Furthermore, the bulk TCP flow alone is able to trigger the th_{max} cutter during its slow start. In the worst case, it is effective for 15.9 seconds, leading to a long period of zero utilization of the link. This is followed by a lengthy period (even up to 100 seconds) with little progress as several retransmissions are dropped, leading to RTOs and multiple RTO backoffs (due to later start time for the burst, the bulk TCP is past these problems when the burst starts).

The elapsed time of the burst over the HSPA link is shown in Figure 4. RED is more effective than with the EDGE when the number of flows is small. HRED outperforms all FIFO and RED variants by a clear margin. Compared to RED+2BDP the median elapsed time is 5.5%-44.5% shorter with HRED. With RED+Large, the th_{max} cutter is effective for 2.0 seconds at worst. HRED+ECN is not decisively better nor worse than HRED, but it suffers from two problems. First, with small number of flows the ECN-marked packets remaining in the queue make the HRED queue average to grow higher during the slow-start phase. The high queue average results in more ECN marks during the following round-trips. These marks lead to multiple TCP window reductions that cause under-

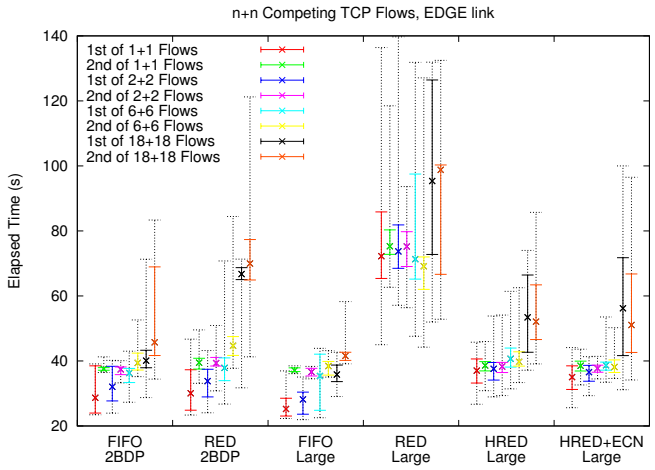

Fig. 5: Elapsed times for the two bursts of TCP flows over an EDGE link

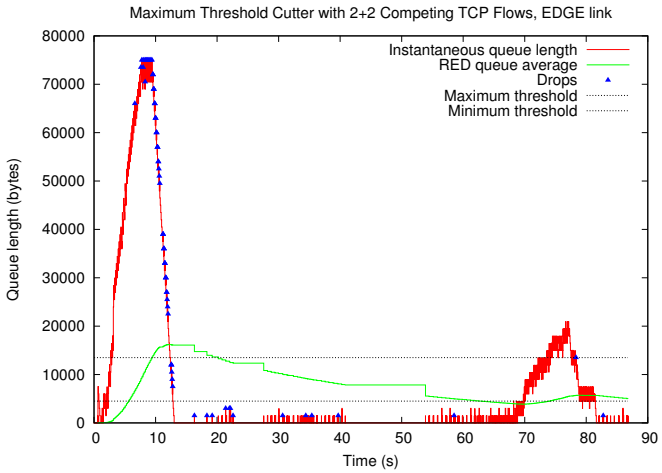

Fig. 6: Queue behavior for 2+2 TCP flows over an EDGE link with RED+Large

utilization making HRED+ECN elapsed times longer than with HRED. Second, with 18 flows some ECN marks are received when the congestion window of the flow is already one, a situation in which the sending rate is reduced further by sending the next new segment only when the retransmission timer expires, delaying the completion of the longest flow occasionally.

The elapsed times for the two bursts with equal number of flows over the EDGE link are shown in Figure 5. 18+18 TCP flows is a total disaster as it simply introduces too high overload for the EDGE link which has BDP less than 7 full-sized packets. Therefore, it is ignored for the rest of the EDGE link analysis. FIFO+2BDP, FIFO+Large and RED+2BDP unfairly favor the first starting burst because of the first comer advantage typical to FIFO behavior, allowing the first TCP burst to complete well in advance compared to the elapsed time of the later TCP burst. As HRED reduces the first comer advantage by introducing timely probabilistic drops, it provides fairness between the flows being superior to all other cases. Median elapsed time for the later burst with HRED is 2.2-25.5% shorter than with RED+2BDP, while HRED has up to 2.8% longer burst elapsed time than FIFO+2BDP. HRED+ECN has 0.2%-4.1% shorter median elapsed time compared to HRED. RED+Large is subject to the th_{max} cutter which almost doubles the elapsed times. An example case with the cutter is presented in Figure 6. Slightly before 10 seconds the queue average crosses th_{max} and all subsequent packets are lost until the average falls below it near 20 seconds. The drops include packets sent with RTO forcing the sender to wait until the exponentially backed off timer expires again. Once the average is within the operating region, the flows can make some progress. However, the congestion windows are small and "pro-active" drops trigger additional RTOs leading to prolonged idle periods. HRED avoids th_{max} cutter and maintains relatively small queue as can be seen in Figure 7.

Figure 8 shows the elapsed times of the TCP bursts over the

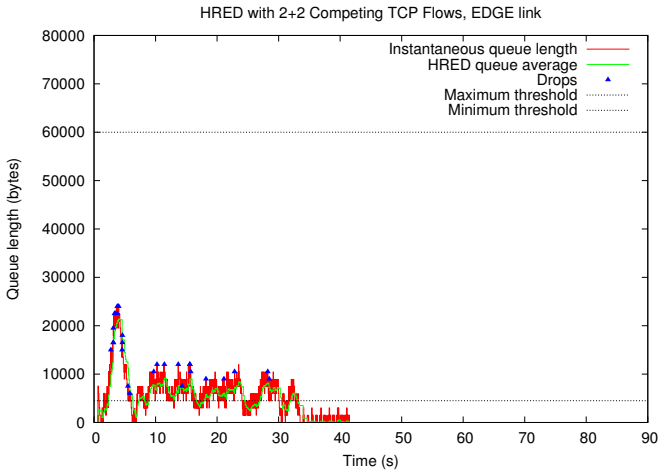

Fig. 7: Queue behavior for 2+2 TCP flows over an EDGE link with HRED

HSPA link. All RED configurations result in longer elapsed times than the FIFO cases because of under-utilization. HRED is the best among the RED alternatives. With RED+2BDP, first the tail-drop fallback occurs as the slow start ends with an slow-start overshoot, as can be seen from the drops in Figure 9. Then during the recovery pro-active drops cause loss of retransmissions which force TCP flows to wait for RTOs. HRED avoids tail-drop and therefore does not encounter slow-start overshoot. Thus the amount of losses during slow start is less than with RED+2BDP. However, lost retransmissions due to pro-active dropping are still common phenomenon causing RTOs also with HRED. When enough RTOs on different TCP flows occur near each other, the link becomes under-utilized and the elapsed times become longer than those with FIFO that only drops when the queue is full. With HRED+ECN a number of RTOs occur when the number of flows is high, i.e., a few RTOs with 6+6 flows and with 18+18 flows the majority of flows encounter RTOs, but their

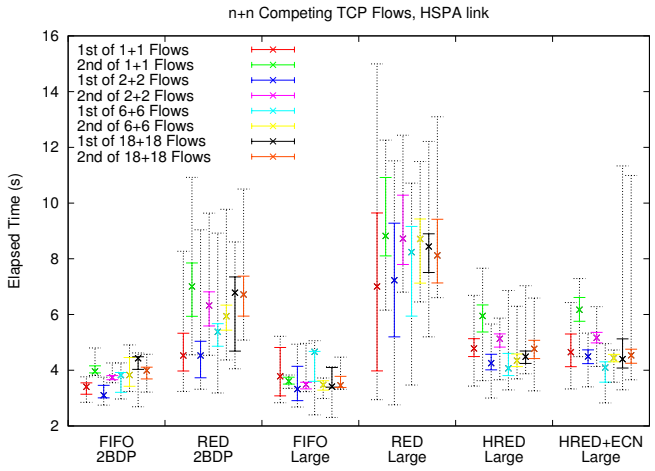

Fig. 8: Elapsed times for the two bursts of TCP flows over an HSPA link

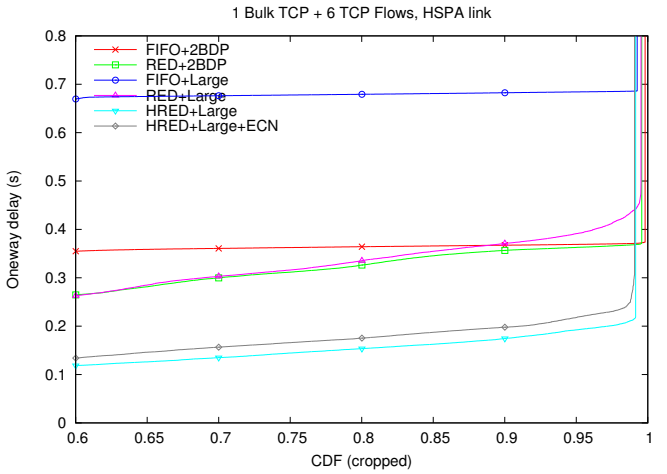

Fig. 10: Oneway delay for a CBR flow competing with a workload of Bulk TCP+6 TCP flows over an HSPA link

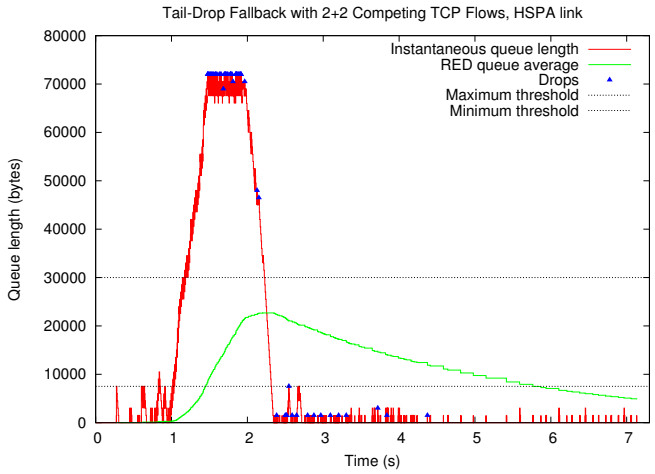

Fig. 9: Queue behavior for 2+2 TCP flows over an HSPA link with RED+2BDP

effect is insignificant for the burst elapsed time. Instead, the burst elapsed time increases because of under-utilization that is caused by ECN markings during consecutive round-trips leading to multiple TCP window reductions. This over-reaction with both HRED and HRED+ECN highlights the EWMA queue average shortcoming; following the initial reaction to slow start HRED marks with a higher probability without giving the flows a chance to react first.

C. Impact on Realtime Traffic

We next focus on the delay effect using workloads that include a CBR flow to measure the delay effect. Figure 10 shows the cumulative distribution function (CDF) for oneway delay that CBR flow experiences when competing with a Bulk TCP+6 flows over an HSPA link. The drop rate is visible in the vertical tail going to infinity. The ”knee” right at the bottom of the vertical drop-rate line indicates the maximum delay experienced by a successfully delivered packet. All queue disciplines achieve less than 1% drop rate. The FIFO cases produce a “flat-line” indicating that the queue remains close to the maximum value. RED+2BDP encounters the buffer limit of 2BDP and reaches roughly the same level as FIFO+2BDP at 366 ms for 15% of the packets. RED+Large diverges to slightly higher delays due to its larger buffer space with the maximum of 494ms. RED+2BDP and RED+Large have only slightly less than 40% of the oneway delays below 200ms. With HRED the worst-case oneway delay is 218ms and the loss rate is 0.8%. Delays above 200ms (or drops) affect 2.8% of the CBR packets with it. HRED+ECN has slightly higher oneway delay and drop rate than HRED because more TCP packets reside in the queue as HRED+ECN marks packets instead of dropping.

With Bulk TCP and 2 flows, all delays are very similar to those with bulk TCP and 6 flows but shift slightly downwards. RED+Large joins RED+2BDP line because the needed buffering is clearly below the available 2BDP-sized buffer. RED+2BDP provides oneway delays that are superior to FIFO+2BDP. However, the oneway delay still remains above 200ms for 25% of the CBR packets and the maximum is 329ms. With RED+2BDP drop rate is 0.1%. HRED has drop rate of 0.6% but keeps the oneway delay below 200 ms with the maximum of 194ms.

Figure 11 shows the CDF for oneway delay with the workload of 2+2 TCP flows over an EDGE link. Again, both FIFO cases have nearly full queue for the highest end of the traffic. RED+2BDP does not provide much lower delays than FIFO+2BDP. However, the drop rate increases from 0.05% to 0.3%. RED+Large is subject to th_{max} cutter causing 34% of the CBR packets to be dropped. With HRED, the drop rate is 0.8% and the delays stay at a lower level than in the RED+2BDP case, except for the 4.9% for the packets at the highest end. With HRED+ECN, the drop rate and oneway delay are again higher than with HRED alone.

Figure 12 illustrates the benefits of HRED from a different

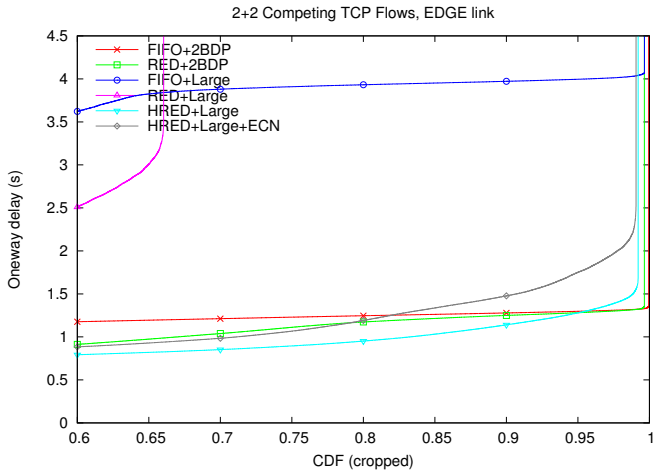

Fig. 11: Oneway delay for a CBR flow competing with a workload of 2+2 TCP flows over an EDGE link

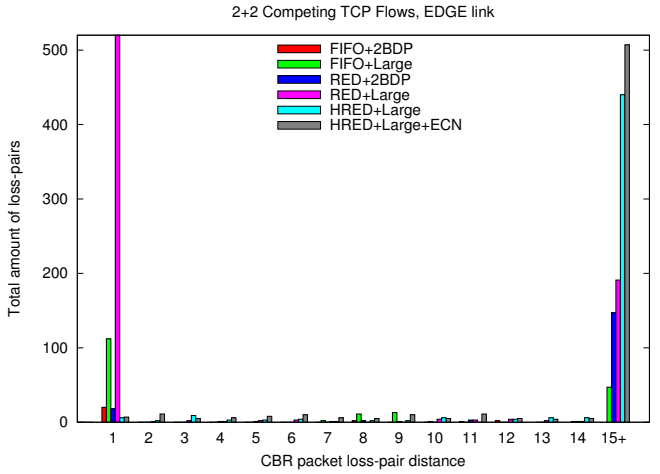

Fig. 12: Loss-pair distance for a CBR flow over an EDGE link (cropped, the maximum with RED+Large for loss-pair distance 1 is 25193)

perspective. This CBR packet-pair loss distance indicates that all other cases have tendency to drop consecutive CBR packets more often than HRED does, whereas with HRED the majority of the drops are distributed well as it is supposed to happen with a properly functioning RED [1].

With 6+6 competing TCP flows over an EDGE link the drop rates again increase; RED+2BDP becomes close to FIFO+2BDP and RED+Large has unacceptable 46% loss rate. With HRED the oneway delays also are longer than with RED+2BDP for 22% of the packets. This is a clear example of overloading the link; there are 12 flows on a link with only 7 packets BDP, i.e., more than one packet per flow will not be bearable in the long term and just adds to the queue length.

The CDF for oneway delays with 6+6 competing TCP flows over an HSPA link are shown in Figure 13. The oneway delays in the highest end follow the patterns explained with the other cases. FIFO+2BDP and RED+2BDP are roughly equal

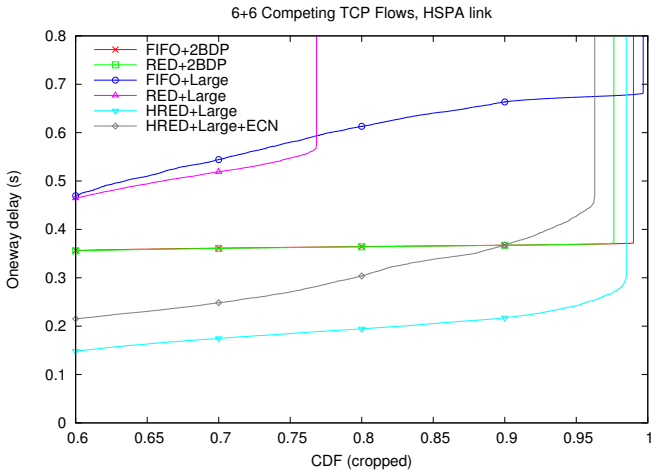

Fig. 13: Oneway delay for a CBR flow competing with a workload of 6+6 TCP flows over an HSPA link

TABLE II: Percentage of CBR packets with oneway delay below 200ms over an HSPA link

	Bulk TCP+n TCP Flows				n + n TCP Flows			
n	1	2	6	18	1	2	6	18
FIFO+2BDP	0%	0%	0%	0%	82%	62%	30%	26%
RED+2BDP	92%	75%	38%	28%	82%	62%	31%	28%
HRED	99.6%	99.4%	97%	82%	94%	90%	83%	78%
HRED+ECN	99.3%	97%	91%	51%	84%	73%	55%	45%

due to the tail-drop fallback. FIFO+Large and RED+Large have long oneway delays and the th_{max} cutter is causing a significant number of drops for the RED+Large case. HRED has superior oneway delay compared to the all other cases. Again HRED+ECN has higher loss rate and longer oneway delay than HRED because of longer queue due to ECN marks instead of drops. The drop rate for RED+2BDP and HRED is 2.3% and 1.5%, respectively. With HRED the maximum delay is 302 ms. The percentage of CBR packets that have oneway delay below 200 ms is summarized in Table II for all workloads over HSPA which confirms that HRED is clearly better than RED+2BDP.

VI. Discussion

During the testing we observed two ns2 RED implementation problems that we fixed before getting the final results. First, setting an equal buffer size for RED and FIFO cause a significant difference in queue size and behavior. With the RED queue, the packet could be queued even if the tail of the packet would not fit the limit as long as the queue was initially below the limit. This bug also raises a question on validity of previous ns2-based simulation studies that compare FIFO and RED. For our workloads the bug caused RED to unfairly favor full-sized TCP packets over small CBR packets. Second, the th_{max} cutter was short-circuited by allowing a packet to be not dropped when instantaneous queue was empty at the arrival of the packet. Therefore, some traffic could pass-through even if the queue average is above th_{max}. Such short-circuit does

not exist in original algorithm [1] nor are we aware of any other work that would mention existence of such behavior. We also confirmed that Linux kernel RED implementation matches here with the implementation given in [1].

With HRED we experimented with nearby weights, but only visible change was shifting of the oneway delays down or up corresponding to a change in queue length. Besides testing HRED with ECN, we also tested both RED+2BDP and RED+Large cases with ECN. In RED+2BDP case the results were nearly identical because pro-active dropping was not that frequent phenomenon, i.e., the tail-drop fallback dominates the behavior with both. With RED+ECN+Large even longer queue was formed because instead of dropping, the packet could be marked and put to the queue. The th_{max} cutter phenomena took even longer time to settle the higher average down causing higher drop rates.

For this study, we intentionally choose a small minimum threshold to allow rapid detection of incipient link saturation to minimize the queue length at the peak before TCP response starts to lower it. A small minimum threshold prevents achieving full utilization if only a small number of flows in TCP congestion avoidance share the bottleneck link. The utilization problem could be mitigated by selecting a larger minimum threshold, however, that would delay the response to the slow start. As the detection of the slow start would be delayed, the queue would be allowed to peak higher which would increase delay. Higher queue peak might also require increasing the maximum threshold and physical queue size. Alternatively, queue average could be made faster by tuning EWMA weight to enable more timely detection.

VII. Conclusions

In this paper we have presented Harsh RED that is intended for links that are subject to limited aggregate traffic. As such it is relevant to a very common case where the bottleneck link is close to the end users. We show that RED with recommended parameters does not work with limited aggregate traffic but falls back to tail-drop or exhibits even worse phenomenon with RED th_{max} cutter. Both of them make RED reactive instead of the intended pro-activeness. We show that with Harsh RED the pro-active behavior is successfully maintained in a number of scenarios typical to end-user behavior. Keeping RED algorithm in pro-active operating region allows shorter oneway delays and better loss distribution. However, we also discovered that RED algorithm has a shortcoming due to which its pro-active correction tends to over-correct during the round-trips following the initial reaction. Such over-reaction leads occasionally to under utilization, especially when the number of flows in the system is small as the window reduction for a single flow has notable effect on the overall load.

The results and problems seem to indicate that making RED more aggressive tends to err to side of caution, instead of making queue average to respond too slowly. Too slow response is subject to both falling back to tail drop and can cause even more dramatic performance drop through the th_{max} cutter phenomenon if the gap between th_{max} and the physical queue size is wide enough. Making RED "too fast" just brings it closer to FIFO behavior (ultimately this would result in what DCTCP [20] does). As random dropping, pro-active response and some burst allowance are still happening, too fast RED does not nullify good features of RED.

Acknowledgments

This work is done in WiBrA project in collaboration with Nokia, Nokia Siemens Networks, and TeliaSonera, funded in part by TEKES Finland.

References

[1] 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.

[2] B. Braden *et al.*, "Recommendations on Queue management and Congestion Avoidance in the Internet," RFC 2309, April 1998.

[3] M. May, J. Bolot, C. Diot, and B. Lyles, "Reasons not to deploy RED," in *Proceedings of IWQoS 2001*.

[4] V. Misra, W. Gong, and D. Towsley, "Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED," in *Proceedings of SIGCOMM 2000*.

[5] M. Christiansen, K. Jeffay, D. Ott, and F. Smith, "Tuning RED for Web Traffic," in *Proceedings of SIGCOMM 2000*.

[6] C. Brandauer *et al.*, "Comparison of Tail Drop and Active Queue Management Performance for bulk-data and Web-like Internet Traffic," in *Proceedings of ISCC 2001*.

[7] L. Le, J. Aikat, K. Jeffay, and F. Smith, "The Effects of Active Queue Management on Web Performance," in *Proceedings of SIGCOMM 2003*.

[8] V. Jacobson, "Congestion Avoidance and Control," in *Proceedings of ACM SIGCOMM '88*.

[9] M. Allman, V. Paxson, and E. Blanton, "TCP congestion control," RFC 5681, Sep. 2009.

[10] S. Floyd, "RED: Discussions of Setting Parameters," Nov. 1997. [Online]. Available: http://www.icir.org/floyd/REDparameters.txt

[11] E. Seurre, P. Savelli, and P. Pietri, *EDGE for Mobile Internet*. Artech House, 2003.

[12] 3GPP, "High Speed Packet Access (HSPA) Release 5 and 6."

[13] V. Jacobson, K. Nichols, and K. Poduri, "RED in a Different Light," unpublished, manuscript, Sep. 1999. [Online]. Available: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.22.9406

[14] T. Ziegler, C. Brandauer, and S. Fdida, "A Quantitative Model for the Parameter Setting of RED with TCP Traffic," in *Proceedings of IWQoS 2001*.

[15] V. Firoiu and M. Borden, "A Study of Active Queue Management for Congestion Control," in *Proceedings of IEEE INFOCOM 2000*.

[16] W. Feng, D. Kandlur, D. Saha, and K. Shin, "A Self-configuring RED Gateway," in *Proceedings of IEEE INFOCOM 1999*.

[17] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management," ICSI Technical Report, 2001.

[18] V. Rosolen, O. Bonaventure, and G. Leduc, "A RED discard strategy for ATM networks and its performance evaluation with TCP/IP traffic," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 3, Jul. 1999.

[19] T. Ott, T. Lakshman, and L. Wong, "SRED: stabilized RED," in *Proceedings of IEEE INFOCOM 1999*.

[20] M. Alizadeh *et al.*, "Data Center TCP (DCTCP)," in *Proceedings of SIGCOMM 2010*.

[21] T. Bonald, M. May, and J. Bolot, "Analytic Evaluation of RED performance," in *Proceedings of IEEE INFOCOM 2000*.

[22] D. Clark, "Window and acknowledgement strategy in TCP," RFC 813, Jul. 1982.

[23] ISI at University of South Carolina, "The network simulator – ns-2." [Online]. Available: http://www.isi.edu/nsnam/ns

[24] K. Ramakrishnan, S. Floyd, and D. Black, "The addition of explicit congestion notification (ECN) to IP," RFC 3168, Sep. 2001.

[25] P. Sarolahti, M. Kojo, K. Yamamoto, and M. Hata, "Forward RTO-Recovery (F-RTO): An algorithm for detecting spurious retransmission timeouts with TCP," RFC 5682, Sep. 2009.