Comparison of Tail Drop and Active Queue Management Performance for bulk-data and Web-like Internet Traffic

Christof Brandauer[1]
Salzburg Research
Salzburg, Austria
cbrand@salzburgresearch.at

Gianluca Iannaccone, Christophe Diot
Sprint ATL
Burlingame, CA, USA
{gianluca,cdiot}@sprintlabs.com

Thomas Ziegler[1]
FTW
Vienna, Austria
ziegler@ftw.at

Serge Fdida
Universite Pierre et Marie Curie
Laboratoire Paris 6, Paris, France
Serge.Fdida@lip6.fr

Martin May
INRIA
Sophia Antipolis, France
Martin.May@activia.net

Abstract

This paper compares the performance of Tail Drop and three different flavors of the RED (Random Early Detection) queue management mechanism: RED with a standard parameter setting, RED with an optimized parameter setting based on a model of RED with TCP flows, and finally a version of RED with a smoother drop function called "gentle RED". Performance is evaluated under various load situations for FTP-like and Web-like flows, respectively. We use measurements and simulations to evaluate the performance of the queue management mechanisms and assess their impact on a set of operator oriented performance metrics. We find that in total (i) no performance improvements of RED compared to Tail Drop can be observed; (ii) fine tuning of RED parameters is not sufficient to cope with undesired RED behavior due to the variability in traffic load; (iii) gentle RED is capable of resolving some of the headaches on RED but not all.

***Keywords**: TCP, RED, performance evaluation*

1. Introduction

Internet best effort traffic is controlled through the interaction of end-to-end congestion control and queue management. TCP's end-to-end congestion control [13, 14] adapts the volume of transmitted data to the current load situation in the net by varying the congestion window as a function of the packet loss rate. Queue management is the decision when to start dropping packets and which packets to drop at a congested router output port. Traditional Internet routers employ "Tail Drop" (TD) queue management, discarding arriving packets if the buffer of the output port overflows. Contrary to TD, active queue management (AQM) mechanisms [10] start dropping packets earlier to enable notification of traffic sources about the *incipient* stages of congestion. Recently, AQM mechanisms [5, 2] have been proposed within the framework of the Internet differentiated services architecture [1] to preferentially drop non conforming over conforming packets.

The Random Early Detection (RED) AQM algorithm [10, 2] has been designed to substitute TD and is nowadays widely implemented in commercially available routers. RED uses the parameter set $\{min_{th}, max_{th}, max_p, w_q\}$ in order to probabilistically drop packets arriving at a router output port. RED maintains an average queue size (avg) which is computed as an exponentially weighted moving average of the instantaneous queue size with weight parameter w_q. If the avg is smaller than min_{th}, no packet is dropped. If $min_{th} < avg < max_{th}$, RED's packet drop probability varies linearly between zero and max_p. If $avg > max_{th}$, the drop probability equals one. Gentle RED (GRED) [9] modifies RED's dropping function for the case that avg exceeds max_{th}, as illustrated in fig. 1. The drop probability increases linearly between max_{th} and the buffer size with a slope of $(1 - max_p)/max_{th}$. Obviously, RED as well as GRED have to drop an arriving packet if the instantaneous queue size equals the total buffer size.

It has been claimed in [10] that RED yields advantages compared to TD mainly in terms of accomodation of bursts and avoidance of global synchronization, i.e. TD's tendency of synchronizing TCP flows to be in phase by dropping packets of several flows at one instance in time. Global syn-

[1]The authors are funded by the IST project Aquila and a research grant from Telecom Austria and FFF Austria

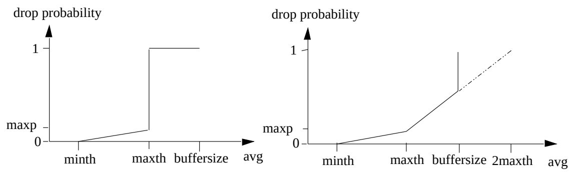


Figure 1. RED and GRED

chronization has the potential to make the queue size oscillate, causing suboptimal throughput and high delay jitter. In order to avoid global synchronization, it has been a design goal of RED to spread out packet losses in time, in other words to minimize the number of consecutive packet drops. It is common knowledge, however, that global synchronization is only a problem in lightly loaded scenarios and infinite length TCP flows [24, 18]. Additionally, suboptimal throughput due to global synchronization can be avoided as long as the buffer size is set to the bandwidth*RTT product of the scenario or higher.

In this paper we evaluate potential benefits of various RED flavors over TD which might give ISPs an incentive to migrate from TD to RED. The rest of this paper is organized as follows: section 2 summarizes related work; section 3 has all the details of the evaluation environment (metrics, testbed and simulation setup, traffic load). Measurement results for FTP-like traffic are discussed in section 4 and simulation results for more realistic Web-like traffic are analyzed in section 5. Finally, section 6 concludes this paper and summarizes the findings.

2. Related Work

As an important starting point for this paper, [19, 12] compare TD, RED and GRED employing standard parameter settings. The main conclusions of these papers are that RED does not improve performance compared to TD, and that tuning of RED parameters is still an open question but should not have a big influence on performance. Similarily, [4] shows that RED exhibits no clear advantages over TD concerning response times of Web transfers and that the setting of RED parameters influences response time performance only in the high load case.

Several publications discussing the setting of RED parameters give either rules of thumb and qualitative recommendations [15, 8, 25] or quantitative models assuming infinite time averages [7] which are not capable of modelling the oscillatory behavior of the RED queue. A control theoretic approach to model the parameter setting of RED is employed in [11].

In [26] a quantitative model how to set RED parameters with TCP traffic is derived. The RED parameters relevant for stability are the maximum drop probability (max_p) determining the equilibrium point (i.e. the queue average over infinite time intervals), the required difference between the queue size thresholds max_{th} and min_{th} to keep the amplitude of the oscillation around the equilibrium point sufficiently low, and the queue weight (w_q). See [3] for a Web interface to the model.

Finally, [27] shows that, independently of the parameter settings, RED-like mechanisms tend to oscillate in the presence of two-way TCP traffic in scenarios lacking heavy cross traffic.

3. Evaluation Environment

3.1. Performance Metrics

We are interested in the aggregate traffic performance of queue management mechanisms and not in individual flow performance or fairness issues. We argue that it makes sense to look at the aggregate traffic behavior first, because earlier papers have already investigated the individual flow performance as well as fairness [20, 16, 4] and second, because ISPs are interested in optimizing aggregate traffic performance. In this paper a traffic mix consisting of TCP flows and constant bit rate UDP flows is used. The total UDP arrival rate equals 10% of the bottleneck link capacity. Similar to [12], the performance metrics are:

TCP goodput is defined as the number of bits per second forwarded to the correct destination interface, minus any bits lost or retransmitted. We decided not to study TCP loss because loss and goodput are somewhat redundant metrics for TCP (TCP adapts its congestion window as a function of the loss rate) and because for TCP flows goodput is of major interest.

UDP loss rate is defined as the number UDP packets that are dropped divided by the total number of UDP packets that arrived at the same output port. For UDP flows the loss rate is an equivalent metric to the goodput for TCP flows. This is an incentive to study aggregate UDP loss which is strongly correlated to the loss rate of individual flows.

Queueing delay: minimizing queueing delay is of interest for transaction-like applications, e.g. remote login or Web transfers. For the testbed measurements the queueing delay is measured inside the dummynet tool as the difference between the time the scheduler starts servicing a packet and the time this packet is enqueued in the output buffer. In order to increase the precision of the measurements, the kernel clock rate in the machine running dummynet is set to 1000 HZ, resulting in a 1 ms granularity for queuing delay.

Standard deviation of queueing delay: queue variance corresponds directly to delay jitter of flows. We decided to measure the standard deviation of the queueing delay instead of the delay jitter of individual flows (i.e. the standard deviation in a flow's interarrival time of packets at the receiver) because delay jitter is not capable of reflecting low frequency oscillations in queueing delay in many scenarios.

Consecutive loss distribution: the mean of the consecutive loss distribution is an important quality criterion for queue management mechanisms as it indicates susceptibility to global synchronization. We define the random variable N as the number of consecutive losses observed at a router output port and consider the probability distribution $P(N > n)$. In the measurements, the dummynet tool measures the loss events which begin when a first packet is dropped and end when the first arriving packet is accepted and enqueued.

The different mechanisms compared in this paper are TD, RED and GRED with a standard parameter setting (REDstd, GREDstd), and RED with an optimal parameter setting (REDopt). REDstd and GREDstd means that min_{th}, max_{th}, and the buffer size are set reasonably high (i.e. in the range of the bandwidth*RTT product of the scenario); max_p is set to 0.1 and w_q is set to 0.002, as recommended in [8]. We define a configuration of RED parameters as optimal (REDopt) if it makes the queue size converge to (min_{th}+max_{th})/2 with only small amplitude oscillations in case of infinite-demand FTP-like flows. Such a queue size behavior is achieved by setting RED parameters according to the model proposed in [26] and summarized in section 2. This model is, however, designed for RED with TCP flows only and not for a mix of TCP plus 10% non-responsive UDP traffic. For this reason we have applied some minor adaptations to the model. The bottleneck link capacity B used as an input to the model is computed as $B - R_{udp} * (1 - Ploss_{udp})$, where R_{udp} denotes the total arrival rate of UDP flows and $Ploss_{udp}$ denotes an estimator for the loss probability of UDP flows. Thus B denotes the fraction of the bottleneck capacity available to the TCP flows. Additionally, as a mix of TCP flows and 10% unresponsive UDP traffic behaves more aggressively than TCP only, we multiply the max_p parameter computed by the model by a factor 1.1 in order to adapt RED's aggressiveness in packet dropping.

3.2. Measurement Environment

The testbed network (fig. 2(a)) used for experiments is meant to represent a gateway where many networks are merged into one outgoing link. All hosts are Pentium PCs running Linux. The PCs have two network cards to enable different paths for incoming and outgoing traffic; this precludes collisions at the link layer. Moreover, PCs 1–4 run a dedicated application to insert additional propagation delay on outgoing traffic to router1. The two routers are Cisco 7500 routers running IOS 12.0.

The bottleneck link, where congestion and packet drops occur, is emulated using dummynet [23], a link emulator that runs on a FreeBSD machine acting as a bridge between the two routers. Dummynet also executes the queue management mechanisms under investigation.

In all experiments the bottleneck link has a capacity of 10 Mbps and a 100 ms propagation delay. The links from PCs 1–4 to router1 have a propagation delay of 20, 40, 60, and 80 ms, respectively. All other links have a negligible propagation delay, so that the end-to-end round trip propagation delay ranges approximately from 120–180 ms.

The MTU size equals 573 bytes for UDP packets and 1514 bytes for TCP packets. Linux implements TCP SACK [17]. Each measurement lasts for 500 seconds, where the initial 100 seconds are not considered in order to filter out the startup behavior of the system; all measurements are repeated 10 times and figures are plotted with 95% confidence intervals.

3.3. Simulation Environment

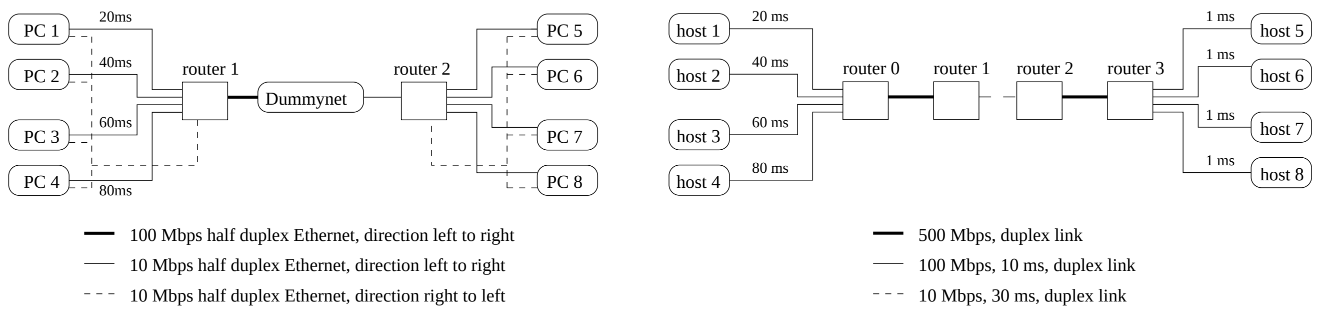
For simulations with the ns-2 simulator [21] we use a topology (fig. 2(b)) similar to the measurement setup. Traffic is multiplexed on the 100 Mbps links between router0 and router1, respectively router3 and router2, before it arrives at the bottleneck link. Output ports transmitting on the 100 Mbps links use TD queue management. Packet drops happen solely at the 10 Mbps link between router1 and router2 where persistent congestion occurs. The queue management algorithm under investigation is executed at the output port of router1. Data senders are located at hosts 1–4. Delays of access links are configured such that the average end-to-end propagation delay equals the end-to-end delay for the measurements. TCP packet sizes are evenly distributed from 1400 to 1628 bytes with a mean of 1514 bytes. UDP packet sizes are set to 573 bytes.

Flows start during the first 10 seconds of simulation time. Simulations last for 5000 seconds; the initial 100 seconds are not considered. With respect to the heavy tailed distributions, all simulations are run 50 times (1000 simulations in total) to get tight 95% confidence intervals as shown in the figures.

3.4. Network Traffic

In each measured and simulated scenario, the background traffic is comprised by four constant bit rate UDP sources that transmit with an aggregate rate of 1 Mbps (i.e. 10% of the bottleneck bandwidth) from hosts 1–4 to hosts 5–8. The foreground traffic is comprised by an aggregate of either FTP-like or Web-like TCP sources as described in the following.

FTP-like TCP flows: In order to investigate the steady state behavior of the different queue management algo-


PC 1
PC 2
PC 3
PC 4
20ms
40ms
60ms
80ms
router 1
Dummynet
router 2
PC 5
PC 6
PC 7
PC 8
100 Mbps half duplex Ethernet, direction left to right
10 Mbps half duplex Ethernet, direction left to right
10 Mbps half duplex Ethernet, direction right to left
host 1
host 2
host 3
host 4
20 ms
40 ms
60 ms
80 ms
router 0
router 1
router 2
router 3
1 ms
host 5
host 6
host 7
host 8
500 Mbps, duplex link
100 Mbps, 10 ms, duplex link
10 Mbps, 30 ms, duplex link

(a) Testbed setup for measurements

(b) Simulated network

Figure 2. Topology of testbed and simulated network

rithms, we analyze traffic from everlasting FTP-like TCP SACK flows. In each set of measurements and simulations the load is varied over a broad range by varying the number of TCP flows. Within the context of window-based congestion controlled flows like TCP, we define the load as the number of flows divided by the bandwidth*RTT product.

Investigations with FTP-like TCP traffic are performed by measurements and simulations. However, the simulation results are not shown in the paper due to space limitations.

Web-like TCP flows: In order to investigate the behavior of the queue management algorithms with more realistic Internet traffic we employ a model for Web traffic developed in [6]. It is a model of HTTP 1.0 traffic which provides several distributions that describe user behavior and properties of Web pages. Please see [6] for the details of the model. The choice of parameters is based on [6] and summarized in table 1.

Our simulations rely on an implementation of the described traffic model that is being shipped with recent distributions of ns-2 [21].

We adapt the load by varying the number of Web sessions. The average number of TCP flows required for the REDopt parameter model [26] is estimated through the average number of active flows observed over the entire simulation. Note that investigations with Web traffic are performed by simulation only.

	Distribution	mean	shape
inter-page time	Pareto	50 ms	2
objects per page	Pareto	4 ms	1.2
inter object time	Pareto	0.5 ms	1.5
object size	Pareto	12 Kbyte	1.2

Table 1. Distributions for Web traffic model

4. Measurements with bulk-data traffic

In order to keep queue oscillation at a constant amplitude, the model [26] used to calculate the parameters for REDopt results in a setting of min_{th}, max_{th} and buffer size (in packets) dependent on the number of flows. Thus, in order to allow for comparison of the queue management mechanisms, the buffer sizes for (G)REDstd and TD are set to the same values as for REDopt. The bandwidth*RTT product of the scenario equals approximately 300 packets (including queueing delay), which is close to the buffer size proposed by the RED model.

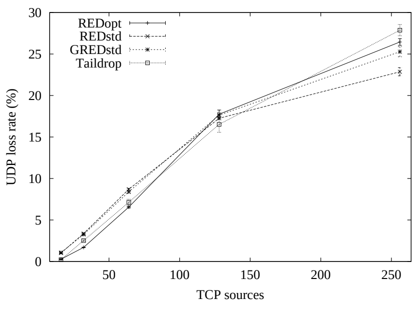
The min_{th} parameter for (G)REDstd is set to 3/20 times buffer size and the max_{th} parameter for (G)REDstd is set to 13/20 times buffer size. The load is varied by changing the number of TCP flows from 16 to 256. The detailed parameter settings are shown in fig. 3(a).

[TCP goodput] The TCP goodput results are practically equal for all queue management mechanisms. In all scenarios goodput is almost optimal (around 8.4 Mbps) as queues hardly ever drain completely and TCP SACK avoids unnecessary retransmissions. In conformance with earlier results, global synchronization does not cause a goodput decrease with TD in lightly loaded scenarios. The goodput is marginally below optimal with GRED and REDstd for the 16 flows scenario as a max_p of 0.1 causes (G)REDstd to drop packets too aggressively for the given flow aggregate. This causes the equilibrium point of the (G)REDstd queue size to stay close to min_{th} which increases the likelihood for the queue to drain completely.

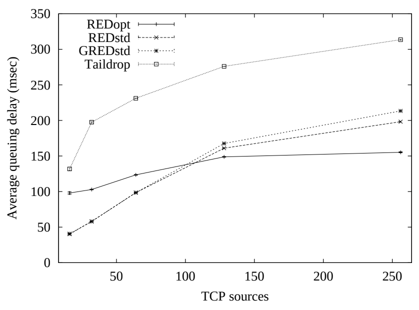
[UDP loss] Fig. 3(b) illustrates that REDopt is better than REDstd and GREDstd by a factor 2 in lightly loaded scenarios concerning the relative difference in UDP loss. TD is in between. In heavily loaded scenarios, the relative differences decrease and are too small to be considered relevant.

	num.flows	minth	maxth	buf.size	maxp	wq
REDopt	16	41	165	247	0.00513	0.00068
	32	42	168	252	0.0185	0.0011
	64	43	172	258	0.0571	0.0016
	128	45	181	272	0.133	0.002
	256	50	200	300	0.233	0.0018
(G)REDstd	16	37	161	247	0.1	0.002
	32	38	164	252	0.1	0.002
	64	39	168	258	0.1	0.002
	128	41	177	272	0.1	0.002
	256	45	195	300	0.1	0.002

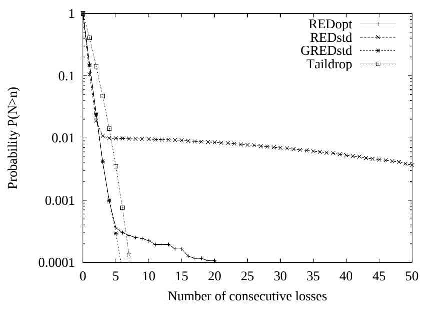
(a) Parameter settings for FTP traffic



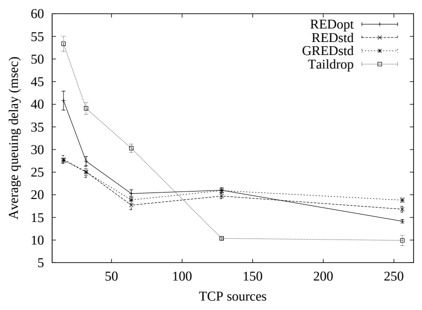
(b) UDP loss



(c) Average queueing delay



(d) Consecutive loss distribution (256 flows)



(e) Standard deviation of queueing delay

Figure 3. Measurements with FTP traffic

[Queueing delay] As the buffer size, max_{th}, and min_{th} parameters are increased directly proportional to the number of flows, queueing delay increases likewise for all mechanisms (see fig. 3(c)). With TD the queue converges to the total buffer size, while the average queue size stays below max_{th} with RED – thus queueing delay is significantly smaller with RED. Note, however, that this can not be considered as an argument against TD. The selection of the buffer size always means balancing a tradeoff between high queueing delay and low link utilization. We have executed simulations showing that TCP goodput, with the TD buffer size set so that TD queueing delay equals the REDopt queueing delay, is the same for TD and REDopt.

Comparing (G)REDstd and REDopt we find that the average queueing delay varies significantly less with REDopt as a function of the number of TCP flows. This is due to the fact that REDopt adapts the max_p parameter to the load, so that the average queue size always stays close to (min_{th}+max_{th})/2. On the contrary, (G)REDstd keeps max_p constant, causing a drift of the average queue from min_{th} to max_{th} as the number of flows increases (see [26]).

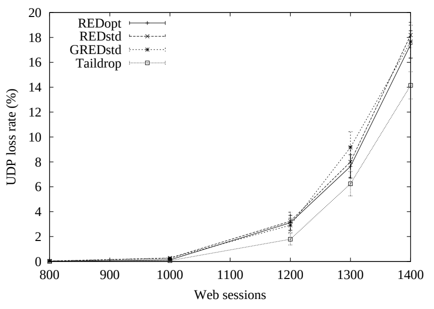
[Delay variation] In case of lightly loaded scenarios the queue size oscillates heavily with TD due to global synchronization, causing a high standard deviation of the queueing delay (see fig. 3(e)). As the load increases, the global synchronization effect disappears; the TD queue stays close to the total buffer size and thus the delay variation decreases significantly.

[Consecutive losses] For lightly to medium loaded scenarios (i.e. less than 64 flows in our case) all RED mechanisms show equal performance concerning the number of consecutive losses. As the average queue size stays below max_{th}, RED is able to buffer traffic bursts without performing consecutive forced drops. TD is significantly worse than the RED variants as the queue size is frequently very close to the total buffer size, enforcing consecutive packet drops in case a burst of packets arrives and causing global synchronization.

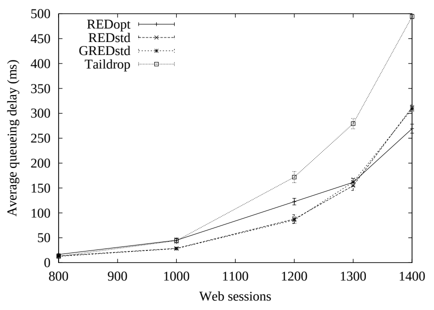
In case of high load (i.e. 256 TCP flows) REDstd performs badly as the average queue size often execeeds max_{th}, enforcing frequent consecutive packet drops (see fig. 3(d)). Due to GREDstd's smoother drop function consecutive drops are less likely with GREDstd than with REDstd. REDopt keeps the average queue size below max_{th} and thus avoids consecutive drops in general. However, from time to time a traffic burst causes the average queue to ap-

	sessions	minth	maxth	buf.size	maxp	wq
REDopt	800	38	151	226	0.00178	0.00008
	1000	38	153	230	0.00472	0.00011
	1200	41	164	246	0.02	0.00021
	1300	49	196	294	0.109	0.00039
	1400	74	299	449	0.333	0.00044
(G)REDstd	800	34	147	226	0.1	0.002
	1000	35	150	230	0.1	0.002
	1200	37	160	246	0.1	0.002
	1300	44	191	294	0.1	0.002
	1400	67	292	449	0.1	0.002

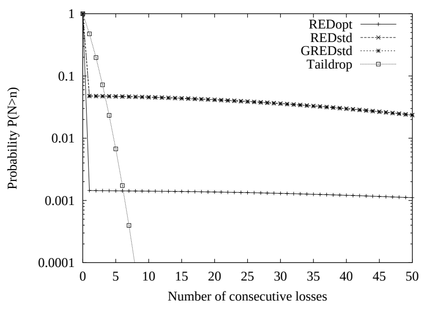
(a) Parameter settings for Web traffic



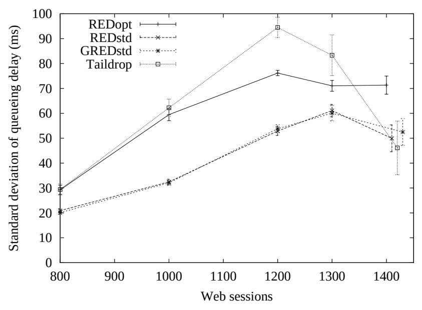
(b) UDP loss



(c) Average queueing delay



(d) Consec. loss distribution (1400 sessions)



(e) Standard deviation of queueing delay

Figure 4. Simulations with Web traffic

proach max_{th}, causing a heavy tail in the REDopt consecutive loss distribution.

5. Simulations with Web-like Flows

While the study of FTP-like traffic was done both with measurements and simulations (showing very good conformance) the study of Web-like flows is restricted merely to simulations. The adaptation of buffer sizes for (G)REDstd and TD is done in the same way as described in the first two paragraphs of section 4. The detailed parameter settings are shown in fig. 4(a).

The load is varied by changing the number of Web sessions from 800 to 1400. Scenarios with fewer sessions than 800 were underloaded and thus not of interest. On the other hand, simulations with more than 1400 sessions led to a situation where the network was extremely overloaded. The flow lifetime became thus very long and the situation converged to the scenarios with FTP-like traffic. A good indicator for the network load is REDopt's max_p parameter which equals 2 times the estimated drop probability of a scenario [26]. As can be seen in fig. 4(a), max_p increases by a factor 2.6 if the number of sessions increases from 800 to 1000, while max_p increases by a factor 16.7 if the number of sessions increases from 1200 to 1400. This shows the highly non-linear behavior of the network load as a function of the number of Web sessions and that the spectrum of load situations where AQM is able to control the system dynamics is very limited.

[TCP goodput] TCP goodput increases monotonically with the number of Web sessions (i.e. the load) and converges to a maximum of 8.7 Mbps for the high load scenarios. The monotonic increase is due to the decreasing probability that the queue is empty because of underload. No significant differences between the queue management algorithms can be found.

[UDP loss] Fig. 4(b) illustrates the non-linearity in system load as a function of the number of Web sessions. The increase of the UDP loss rate curves (indicating network load) over the number of sessions is significantly non-linear and completely contrary to the loss rate curves for FTP traffic (see fig. 3(b)). TD generally has a lower loss rate, especially for 1200 Web sessions; the other scenarios show only minor differences.

[Queueing delay] As shown in fig. 4(c), the infinite-time average of the queueing delay increases proportionally with the number of Web sessions for all mechanisms because the queue size thresholds determined by the model for REDopt increase likewise. As REDopt adapts max_p to the network load, it is best in making the infinite-time average of the queue size converge close to (min_{th}+max_{th})/2, causing the average queueing delay to increase only moderately as a function of the number of Web sessions. All other mechansisms are not able to control the infinite-time average of the queue size, causing the steep increase of the queueing delay. For TD, the average queueing delay is generally higher than for RED because TD's average queue has the tendency to converge to the buffer size while for RED it does not exceed max_{th} (see section 4).

[Delay variation] In case of Web traffic the standard deviation of the queueing delay is mainly determined by the burstiness of the arriving traffic in medium load scenarios. The performance differences are partly due to the different ranges of queue size variation allowed by the mechanisms. For instance, TD allows variation between zero and the entire buffer size while REDstd allows variation between zero and max_{th}, thus REDstd generally has a smaller standard deviation of queueing delay than TD. Similar to scenarios with FTP traffic the standard deviation of TD decreases if the load increases because the queue size exhibits a tendency towards convergence to the total buffer size.

Comparing (G)REDstd and REDopt we observe that REDopt has a higher standard deviation, independently of the settings of max_p (for low load scenarios REDopt has a smaller max_p than REDstd, vice versa for high load scenarios). The settings for min_{th} and max_{th} are approximately the same. Thus, as the queue weight is the only remaining parameter, the longer memory in REDopt's queue size averaging (REDopt has a smaller queue weight than REDstd) seems to have a negative effect on the standard deviation of the queueing delay with Web traffic.

[Consecutive Losses] Fig. 4(d) shows that the RED variants are significantly worse than TD concerning consecutive drops. While REDopt is able to keep the queue size below max_{th} for FTP traffic (see section 4) and thus avoids consecutive losses, it fails to do so for the Web traffic. Due to the high variability and burstiness of Web traffic the average queue size (of all RED variants) frequently approaches max_{th}, causing consecutive losses.

We looked at different load situations and took into account the different settings of max_p and w_q of the mechanisms under investigation. From these observations it becomes obvious that a higher value for w_q (i.e. shorter memory in the queue averaging) decreases the probability for long periods of consecutive drops. This is due to the fact that once the average queue size is close to max_{th}, it will stay there longer the more history from the averaging process (low value for w_q) is taken into account.

6. Conclusions

We have compared the performance of Tail Drop (TD), RED and GRED with a standard parameter setting (REDstd, GREDstd) and RED with fine-tuned parameter setting (REDopt). Performance metrics are TCP goodput, UDP loss rate, average queueing delay, standard deviation of the queuing delay (as an indicator for delay jitter) and the consecutive loss probability distribution. The analysis is performed with a traffic mix consisting of FTP-like TCP flows and 10% of UDP traffic, as well as a traffic mix consisting of Web-like TCP flows and 10% UDP traffic for various load situations.

Our main findings are:

- the mechanisms do not show significant differences concerning TCP goodput and UDP loss.
- In all load situations for FTP and Web traffic, REDopt is superior in controlling the average queueing delay.
- TD exhibits higher variation in the standard deviation of the queueing delay as a function of the load than the RED variants. Moreover, we observe that longer memory in RED's computation of the average queue size increases the standard deviation of queueing delay for Web-like flows.
- GREDstd and REDopt are significantly better than REDstd concerning consecutive losses for medium to high load scenarios in case of FTP traffic. In these scenarios the RED queues stay close to the max_{th} threshold, frequently causing consecutive "forced" packet drops. In case of Web traffic, all AQM mechanisms fail to keep the average queue size away from max_{th} due to the high variability in traffic load. Consequently, these mechanisms perform badly in terms of consecutive losses. On the other side, TD performs well concerning consecutive losses, quite independently of the load situation for FTP and Web traffic. Note that these kinds of forced consecutive packet drops with RED cause mechanisms like ECN [22] to fail.

Basically, the performance of REDstd is sensitive to the load situation for most of the used metrics, which can be considered as a serious drawback. REDopt is less sensitive to the load situation as its parameters are adapted based on a model assuming knowledge of the network scenario (bottleneck bandwidth and RTT) and the load situation. However, in realistic scenarios with Web traffic the load situation (e.g. number of TCP flows and their demand) varies heavily, thus REDopt is not capable of resolving the drawbacks inherent to RED. However, models as proposed in [7, 11, 26] provide important insights on how to set the parameters of optimized AQM mechanisms (e.g. GRED) in the future and especially on the reasons behind the behavior of AQM mechanisms.

TD exhibits significant dependency on traffic variability concerning average queueing delay and its standard deviation. In lightly loaded scenarios with FTP traffic TD frequently drops packets within a short time-window, causing global synchronization. Our performance evaluation shows that global synchronization causes a high standard deviation of queueing delay, but the effect on TCP goodput is negligible. Additionally, for Web traffic and for medium to high load scenarios with FTP traffic, global synchronization disappears with TD and the standard deviation of the queueing delay decreases in case the load increases.

In total, our evaluation does not show superiority of RED over TD. Thus ISPs currently do not have an incentive to migrate from TD to RED, given the complexity of RED parameter setting. GRED performs reasonably well in all scenarios but queueing delay and the standard deviation of the queueing delay still depend on the load. Generally speaking, all queue management mechanisms perform well in certain scenarios but sub-optimal in others. Thus the development of new active queue management mechanisms can still be considered as a challenge for the research community.

References

[1] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. RFC 2475: An architecture for differentiated services, Dec. 1998.

[2] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. RFC 2309: Recommendations on queue management and congestion avoidance in the Internet, Apr. 1998.

[3] C. Brandauer. Web interface to the RED model developed in [26], 2000. `http://www.salzburgresearch.at/~cbrand/REDmodel`.

[4] M. Christiansen, K. Jeffay, D. Ott, and F. Smith. Tuning RED for web traffic. In *Proceedings of ACM SIGCOMM 2000, Stockholm, Sweden*, pages 139–150, Sept. 2000.

[5] D. Clark and W. Fang. Explicit allocation of best effort packet delivery service. *IEEE/ACM Transactions on Networking*, 6(4):362–373, 1998.

[6] A. Feldmann, A. C. Gilbert, P. Huang, and W. Willinger. Dynamics of IP traffic: A study of the role of variability and the impact of control. In *Proceedings of ACM SIGCOMM '99*, pages 301–313, 1999.

[7] V. Firoiu and M. Borden. A study of active queue management for congestion control. In *Proceedings of IEEE Infocom, Tel Avivthe 2000 IEEE Computer and Communications Societies Conference on Computer Communications (INFOCOM-00)*, pages 1435–1444, Mar. 2000.

[8] S. Floyd. Discussions on setting RED parameters, Nov. 1997. `http://www.aciri.org/floyd/REDparameters.txt`.

[9] S. Floyd. Recommendations on using the gentle_ variant of RED, Mar. 2000. `http://www.aciri.org/floyd/red/gentle.html`.

[10] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, Aug. 1993.

[11] C. Hollot, V. Misra, D. Towlsey, and W. Gong. A control theoretic analysis of RED. In *Proceedings of the IEEE Infocom 2001, to appear*, 2001.

[12] G. Iannaccone, M. May, and C. Diot. Aggregate traffic performance with active queue management and drop from tail. Technical Report TR01-ATL-012501, Sprint ATL, Jan. 2001.

[13] V. Jacobson. Congestion Avoidance and Control. *Computer Communication Review*, 18(4):314–329, Aug. 1988. Proceedings of the Sigcomm '88 Symposium in Stanford, CA, August, 1988.

[14] V. Jacobson. Modified TCP Congestion Avoidance Algorithm, Apr. 1990. end2end-interest mailing list.

[15] V. Jacobson, K. Nichols, and K. Poduri. RED in a different light. Technical report, Sept. 1999.

[16] D. Lin and R. Morris. Dynamics of random early detection. In *Proc. SIGCOMM 97 Conference*, 1997.

[17] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. RFC 2018: TCP selective acknowledgment options, Oct. 1996.

[18] M. May. *Performance Evaluation of recent/new QoS Mechanisms in the Internet.* PhD thesis, Université de Nice, Sophia Antipolis, 1999.

[19] M. May, J. Bolot, C. Diot, and B. Lyles. Reasons not to deploy RED. In *Proc. of 7th. International Workshop on Quality of Service (IWQoS'99), London*, pages 260–262, June 1999.

[20] R. Morris. TCP behavior with many flows. In *IEEE International Conference on Network Protocols*, Oct. 1997.

[21] Network Simulator ns-2, see `http://www.isi.edu/nsnam/ns/`.

[22] K. Ramakrishnan and S. Floyd. RFC 2481: A proposal to add Explicit Congestion Notification (ECN) to IP, Jan. 1999.

[23] L. Rizzo. Dummynet: a simple approach to the evaluation on network protocols. *ACM Computer Communication Review*, Jan. 1997.

[24] T.J.Ott, T.V.Lakshman, and L.Wong. SRED: Stabilized RED. In *Proc. IEEE Infocom, San Francisco, CA*, Mar. 1999.

[25] C. Villamizar and C. Song. High performance TCP in ANSNET. *Computer Communication Review*, 24(5):45–60, 1994.

[26] T. Ziegler, C. Brandauer, and S. Fdida. A quantitive model for parameter setting of RED with TCP traffic. In *Proceedings of the Ninth International Workshop on Quality of Service (IWQoS), 2001, Karlsruhe, Germany, to appear*, 2001.

[27] T. Ziegler, S. Fdida, and C. Brandauer. Stability of RED with two-way TCP traffic. In *IEEE ICCCN, Las Vegas*, Oct. 2000.