# Deficit Round Robin with Limited Deficit Savings (DRR-LDS) for Fairness among TCP Users

Michael Menth, Marcel Mehl, and Sebastian Veith

University of Tuebingen, Chair of Communication Networks,
Sand 13, 72076 Tuebingen, Germany
{menth@,marcel.mehl@student.,sebastian.veith@}uni-tuebingen.de

**Abstract.** Deficit Round Robin (DRR) is a simple and computationally efficient approximation of the Weighted Fair Queueing (WFQ) scheduling discipline. Its intention is to share resources among several queues, e.g., flows or users, according to given weights. However, when users hold different numbers of TCP connections with saturated sources, the throughput among these users may differ significantly.
In this work, we quantify the difference in throughput for heavy and light users with saturated TCP flows for equal weights and for two different buffer management strategies. The difference is large if low queueing delay for packets is enforced through shallow buffers on the bottleneck link. To address this shortcoming, we propose limited deficit savings (LDS), a modification of the DRR scheduler, which can be combined with different buffer management schemes. We show that LDS reduces unequal throughput for heavy and light users with saturated TCP flows. Moreover, we illustrate that LDS clearly decreases download times for data chunks of moderate size in the presence of high background load.

**Keywords:** Congestion management, scheduling, fairness, buffer management

## 1 Introduction

Deficit Round Robin (DRR) [28] is a computationally efficient approximation of the Weighted Fair Queuing (WFQ) scheduler. It serves several packet queues and allocates to them the capacity of a single server, e.g., the bandwidth of a communication link, according to configurable weights. In particular, DRR respects packet sizes so that queues cannot obtain larger capacity shares by sending larger packets.

While fair allocation of transmission bandwidth has been the focus for many years, low delay has recently become more important. Drivers for low-delay

The authors acknowledge the funding by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/2-1. The authors alone are responsible for the content of the paper.

transmission in the Internet and communication networks in general are real-time applications like voice over IP, video conferencing, financial applications, and gaming. Several specialized working groups in the Internet Engineering Task Force (IETF) work on mechanisms to reduce delay and congestion in the Internet: ConEx [16], RMCAT [17], and AQM [15]. The FP7 project RITE pursues that objective [1] and the Internet Society organized two workshops in this area [18, 19]. An overview to reduce Internet latency is given in [5].

If low packet delay is desired with WFQ or DRR, some packets are dropped if the buffer holds too many of them. Such a drop decision is part of the buffer management schemes. We basically consider two strategies: drop-on-enqueue and drop-on-dequeue. As TCP is still the predominant transport protocol in the Internet, we investigate the transmission of saturated TCP sources. Heavy users may have more TCP connections than light users. We show that both considered buffer management methods do not lead to equal bandwidth allocation among heavy and light users if low delay is enforced.

A major reason for that phenomenon is the fact that in the DRR algorithm a queue does not benefit in the resource allocation process while it is empty. This problem increases with tighter delay requirements. To mitigate that problem, we propose limited deficit savings (LDS) as an extension to DRR so that queues can collect credits during short periods of inactivity between the last packet sent and the next packet arrived. We evaluated this mechanism under various conditions. To that end, we implemented variants of DRR in the INET simulation framework of OMNeT++ [30] and performed multiple experiments.

The paper is structured as follows. Section 2 explains the DRR algorithm, reviews existing work about DRR and WFQ, active queue management (AQM), and distinguishes our approach from other activities. Section 3 introduces the LDS extensions for DRR. Section 4 explains the experimental setup and discusses simulation results. Section 5 summarizes this work and draws conclusions.

## 2 Related Work

In this section, we briefly introduce WFQ and some of its variants and explain the DRR algorithm. We introduce the notion of bufferbloat, point out several AQM methods, and distinguish these efforts from our approach.

### 2.1 Weighted Fair Queueing and Variants

In 1985, John Nagle discussed the benefits of a round robin scheduling system [23] which was later called fair queueing (FQ). The approach was enhanced by Demers et al. [6] and by Zhang [36] towards a logical bit-by-bit fair scheduler which became known as Weighted Fair Queueing (WFQ). The introduction of weights allows for unequal resource allocation of capacity to different queues. In 1995, Shreedhar and Varghese proposed the Deficit Round Robin (DRR) scheduler [28], which approximates WFQ but is computationally less demanding. Other improved approximations followed like the Worst-case Fair Weighted

Fair Queueing (WF2Q) which utilizes the start time of packets additionally to the finish time to enhance accuracy, or WF2Q+ which improved accuracy and reduced complexity [4].

### 2.2 DRR Algorithm

We now describe DRR in more detail as it is the base for our study. With DRR, queues are associated with weights and the objective of the DRR is to assign the capacity of a server or bandwidth of a link to the active queues in the system according to these weights. A queue is active if it stores a packet, otherwise it is inactive. The queues are administered in an active list and a set of inactive queues. If a packet arrives for an inactive queue, the queue is removed from the set of inactive queues and appended to the end of the active list. The active queues are served in the following manner. Every queue is associated with a deficit counter. The deficit counter of the first queue from the active list is incremented by a quantum, which is an amount of bytes, multiplied by the weight associated with that queue. If the deficit counter is at least as large as the size of the first packet in the queue, the deficit counter is decreased by the size of that packet, and that packet will be sent next. This process continues until the deficit counter is not large enough to send the next packet or until the queue is empty. The queue is then removed from the active list. If it still holds packets, it is appended again to the end of the active list, otherwise it is added to the inactive set and the deficit counter is reset to zero. Then the process of assigning a quantum to the first queue in the active list and sending packets continues. DRR uses a shared buffer for all queues. If there is no space left in the buffer upon arrival of a new packet, a packet of the longest queue is dropped. This buffer management is called McKenney's buffer stealing algorithm [22]. In the following we refer to it as drop-on-enqueue.

### 2.3 Bufferbloat

Sufficiently large buffers are needed under certain conditions to achieve good bandwidth utilization on networking hardware [7, 8], but if they are filled for relatively long time, packets experience excessive and unnecessary delay. This phenomenon is called bufferbloat [10], i.e., excessive packet delay due to large and unmanaged buffers. The general problem behind bufferbloat was already described in 1985 by John Nagle [23]. Quantitative evaluations showed that bufferbloat is not ubiquitous and its impact may be limited [3, 14]. Bufferbloat in cellular networks has been studied in [20].

### 2.4 Active Queue Management

Active queue management (AQM) is a class of buffer management schemes that counteract incipient congestion by dropping or marking packets before the queue is fully occupied. A classic scheme is Random Early Detection (RED) [9] that

drops packets with a probability rising with the recent average queue occupation. With explicit congestion notification (ECN), packets are rather marked instead of dropped, but ECN is applicable only if TCP sources indicate to reduce their traffic rate in response to appropriate congestion signals from TCP receivers [27]. The PIE controller is an enhanced AQM whose packet drop probability depends on growing and shrinking average queue sizes [26]. It is already applied for DOCSIS systems [35] [33] [34, Annex M]. The Controlled Delay (CoDel) AQM [24,25] is currently proposed as a countermeasure against bufferbloat and already integrated in many stacks. It is extended towards FQ-CoDel [13] by combining it with DRR and Stochastic Fair Queueing (SFQ) [22] so that low-rate flows within a traffic aggregate do not suffer excessive delay due to competing high-rate flows. Various new AQMs have been compared in [21], and [2] reviews a multitude of AQMs that have been discussed in the past. Interactions between AQMs and low-priority congestion control have been investigated in [11,12]. The authors of [29] have considered various buffer management strategies together with per-flow queueing strategies in Gigabit routers.

### 2.5 Our Approach

In our work we consider a scheduler with multiple queues, one for a specific aggregate that we call a user in the following. Each user may have multiple flows and the objective of the scheduler is to enforce a resource allocation to queues according to configured weights. This is typically achieved by WFQ. However, another goal is to keep packet delay low which is the objective of AQM or buffer management mechanisms. We try to achieve low delay with DRR with simple buffer management strategies and to understand their impact. FQ-CoDel is close to our approach in the sense that it keeps delays low and uses DRR, but it uses SFQ to isolate flows of a single user in different queues. Of course, CoDel could be easily modified for per-user queueing. The objective of our work is the enhancement of DRR by LDS from which other mechanisms like FQ-CoDel could also profit.

## 3 Limited Deficit Savings for DRR

With DRR, the deficit counters of queues are increased only if they hold at least a single packet; otherwise they are not respected for the resource allocation process. However, when low latency is enforced in the presence of multiple queues, queues are likely to hold no packet most of the time even though the system is fully utilized. As a consequence, queues with less frequent packet arrivals are disadvantaged in competing for the capacity share given by the weights. This problem arises for TCP traffic.

Our idea is to allow an empty queue to increase its deficit counter up to a deficit saving limit $D_{max}$. At the next packet arrival, the queue does not need to wait for a deficit increase but can be served preferentially within a set of queues that have enough deficit to send packets. We describe this DDR-LDS algorithm in the following.

### 3.1 State Classification of Queues

A queue is fresh if it is empty and if its deficit counter $D$ equals $D_{max}$. All fresh queues are kept in the "fresh list" $\mathcal{F}$. A queue is eligible if it holds at least one packet and if its deficit counter $D$ is at least as large as the size $B$ of its first packet so that this packet could be sent. All eligible queues are kept in the "eligible list" $\mathcal{E}$. An empty queue is collecting if its deficit counter $D$ is smaller than $D_{max}$. A non-empty queue is collecting if its deficit counter $D$ is smaller than the size of its first packet $B$ so that this packet cannot be sent. All collecting queues are kept in the "collecting list" $\mathcal{C}$. Figure 1 depicts how queues move from one list to another if their state changes. The states of the queues in the respective lists are indicated in the circles. The algorithms in the following describe how these state changes are triggered and when the queues are moved.

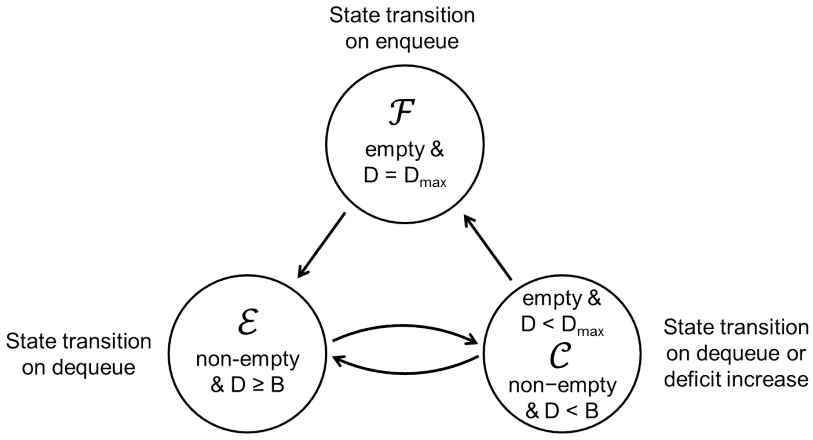



**Fig. 1.** A queue belongs to the fresh, eligible, or collecting list depending on its state. Queues are moved from one list to another during the execution of DRR-LDS.

### 3.2 Packet Enqueue

If a packet arrives, it is appended to its queue. If the queue was fresh before packet arrival, it is removed from $\mathcal{F}$ and appended to $\mathcal{E}$. If the queue was empty and in $\mathcal{C}$ before, it may need to be removed from $\mathcal{C}$ and appended to $\mathcal{E}$, depending on its new state. If the link is idle after a packet arrival, a packet may be immediately dequeued for transmission as described in the next paragraph.

### 3.3 Packet Dequeue

If a packet is needed for transmission and $\mathcal{E}$ is not empty, the algorithm removes the first packet of the first queue in $\mathcal{E}$. If that packet will be dropped for some

reason and if the queue is empty afterwards, the queue is removed from $\mathcal{E}$ and appended to $\mathcal{C}$ or $\mathcal{F}$ depending on the queue classification criteria. If the packet will be transmitted, the queue is removed from $\mathcal{E}$, its deficit counter is decreased by the packet size, and the queue is appended to $\mathcal{E}$ or $\mathcal{C}$, depending on its new state. This procedure repeats until a packet for transmission is found or until $\mathcal{E}$ is empty. In the latter case the deficit will be increased as described in the next paragraph.

### 3.4 Deficit Increase

If $\mathcal{E}$ is empty, the deficit counters of queues in $\mathcal{C}$ are increased until $\mathcal{E}$ is no longer empty or until $\mathcal{C}$ itself is empty. To that end, the deficit counter of the first queue in $\mathcal{C}$ is increased by the quantum multiplied by the queue's weight, but the deficit counter cannot exceed $D_{max}$ if the queue is empty. Then, the queue is removed from $\mathcal{C}$ and then appended to $\mathcal{F}$, $\mathcal{E}$, or $\mathcal{C}$. This procedure repeats until $\mathcal{C}$ is empty or until $\mathcal{E}$ becomes non-empty. In the latter case, packet dequeue is resumed.

### 3.5 Some Observations

Queues can collect deficit only if no queue is eligible. If the buffer runs empty, all queues become fresh again. FQ-CoDel implements a similar mechanism that is described in [13]. However, with FQ-CoDel the time empty queues take to become fresh again is independent of $D_{max}$. With DRR-LDS the time an empty queue needs to become fresh again does depend on $D_{max}$.

## 4 Performance Evaluation

We first describe the simulation setup. Then we show that DRR cannot enforce equal bandwidth allocation in the presence of heavy and light users with TCP traffic and low latency requirements. We demonstrate that DRR-LDS reduces this unequal bandwidth allocation and leads to fast downloads of short transactions for certain buffer management schemes.

### 4.1 Experiment Setup

We implemented DRR and DRR-LDS in combination with two buffer management strategies based on the INET framework [30] for the discrete-event simulator OMNeT++ [31]. As we do not trust INET's TCP implementation, we use the Network Simulation Cradle [32] based on which INET allows to integrate Linux networking stacks for TCP New Reno and Cubic.

Figure 2 illustrates our simulation setup. A set of "users" is connected to one router over a fast link with a bandwidth of $C_a = 1$ Gb/s and a one-way propagation delay of $D_a = 1$ $\mu$s. This router connects to a server over a slow bottleneck link with a bandwidth of $C_b = 10$ Mb/s and a one-way propagation delay of $D_b = 5$ ms or $D_b = 50$ ms, respectively. Thus, the transmission time of a packet

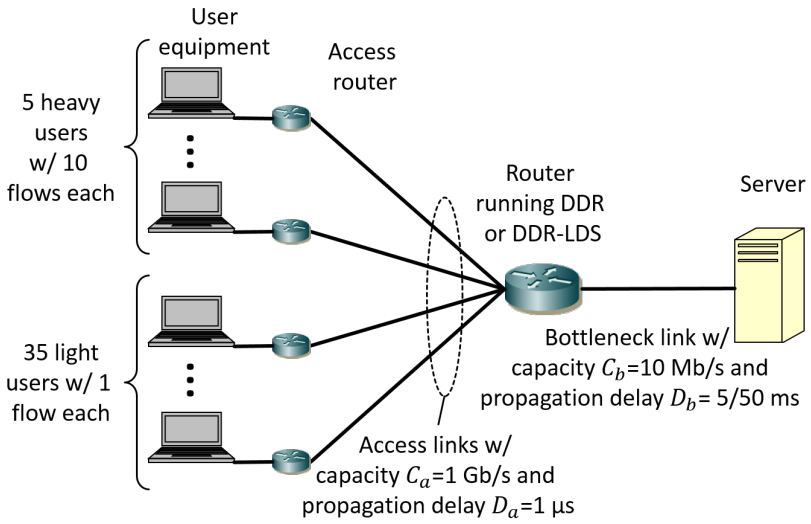



**Fig. 2.** Simulation setup.

with an average size of 1500 bytes takes 1.2 ms. We apply DRR or DRR-LCS for this bottleneck link and assign equal weights to all users. We configured the DRR with a quantum of 1500 bytes. The experiments use saturated TCP connections, i.e., there is always data to send. The TCP connections start randomly within the first second of a simulation run and statistic collection starts only after 5 s to avoid transient effects. If not mentioned differently, each simulation run is repeated 100 times.

### 4.2 Buffer Management Schemes

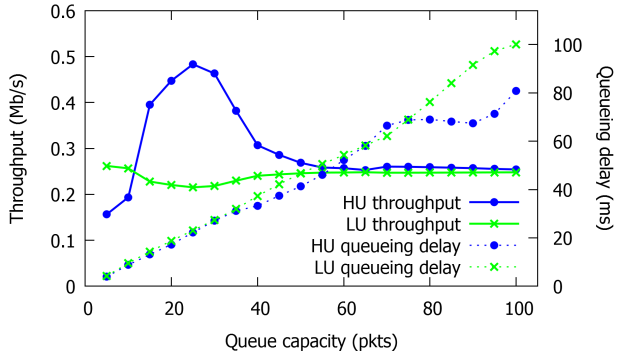
We consider two buffer management schemes for a shared buffer: *drop-on-enqueue* and *drop-on-dequeue*.

Drop-on-enqueue is the strategy originally proposed with DRR: if a packet arrives and the buffer is fully occupied, the oldest packet of the longest queue is dropped. Thereby, the packet delay can be controlled by the buffer size $S_B$.

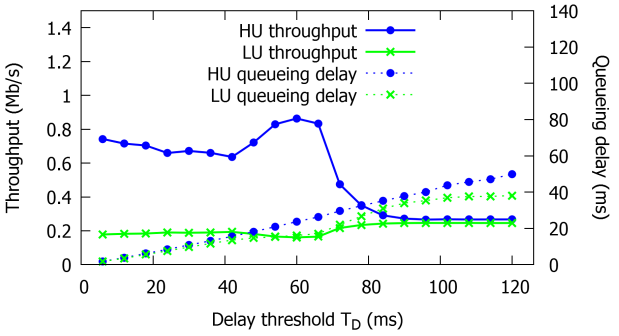
Drop-on-dequeue is inspired by new AQMs like CoDel, but we pursue a very simple approach. A packet is dropped if it is older than a configurable delay threshold $T_D$. This method limits the packet delay to $T_D$ and removes packets from the queue in case of congestion. Nevertheless, the buffer can overflow under certain conditions. To avoid that, we postulated a sufficiently large buffer and assumed infinite for the sake of simplicity.

### 4.3 Resource Allocation with DRR

The objective of this experiment is to test whether DRR can enforce an intended resource allocation in the presence of many users so that DRR queues run empty. We consider $n = 40$ active users, 5 heavy users holding 10 TCP connections with the server and 35 light users holding only a single TCP connection with the server. With FIFO scheduling, we expect ratios of 10:1 for achieved transmission rates of heavy and light users assuming TCP's per-flow fairness is perfect. With DRR scheduling we expect equal transmission rates of 0.25 Mb/s for heavy and light users.




(a) Drop-on-enqueue.




(b) Drop-on-dequeue.

**Fig. 3.** Throughput and packet queueing delay for heavy users (HU) and light users (LU). Results are shown for TCP New Reno and $D_b = 5$ ms.

Figure 3(a) shows the throughput and packet queueing delay of heavy and light users with TCP New Reno connections for drop-on-enqueue buffer management and with $D_b = 5$ ms on the bottleneck link. The x-axis shows the

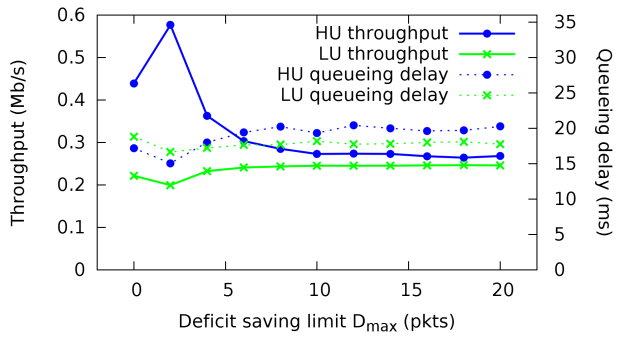
buffer size and the y-axes the throughput and packet queueing delay of each user type. For buffer sizes between 15 and 40 packets, heavy users achieve significantly larger transmission rates than light users. This is undesired insofar as both heavy and light users have saturated sources an should share the link equally. The queueing delay for packets rises about linearly with the buffer size and its mean reveals that about $\frac{5}{6}$ of the queue is occupied on average. Thus, the queue is often filled which is due to the relatively high load. For very large queue capacity, we observe shorter delays for heavy users than for light users. As DRR with drop-on-enqueue requires 50 packets to achieve equal bandwidth allocation for competing TCP users in our setting, it cannot enforce both fairness and low or moderate delay under heavy load.

Figure 3(b) presents similar results for drop-on-dequeue. The x-axis shows the delay threshold $T_D$ instead of the buffer size $S_B$; the range [6;120] ms is chosen because that corresponds to the transmission times of a packet range [5;100] which was used for drop-on-enqueue in Figure 3(a). For drop-on-dequeue we observe even larger differences in throughput for heavy and light users, in particular for small delay thresholds. A delay threshold of $T_D = 90$ ms is needed to achieve equal bandwidth allocation for heavy and light users which results in almost 40 ms average packet delay for both user types. With drop-on-dequeue, packet queueing delay rises about linearly with the delay threshold $T_D$ and is similar for heavy and for light users due to the absence of buffer stealing. It is about half the delay threshold $T_D$ in our experiments. With drop-on-enqueue, the queue length cannot exceed $S_B$ but the buffer was mostly fully occupied, which is not shown in the figures. For drop-on-dequeue the buffer size was not limited. Nevertheless, the measured average queueing delay linearly increased from zero (for $T_D = 0$ ms) to 50 ms (for $T_D = 120$ ms). Thus, the average queueing delay for drop-on-dequeue is significantly lower than the corresponding average queueing delay for drop-on-enqueue.

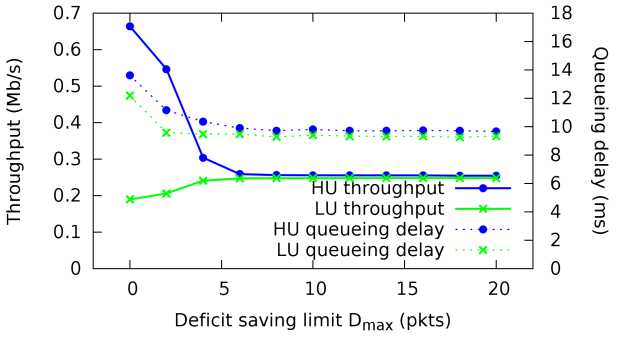
We performed the same experiments with $D_b = 50$ ms and obtained almost the same results. For TCP Cubic we received different results but the same conclusion: large queue capacity is needed for fair bandwidth sharing.

### 4.4 Resource Allocation with DRR-LDS

We evaluate the effect of limited deficit savings (LDS) on the throughput of heavy and light users. We first consider drop-on-enqueue for a buffer size of $S_B = 20$ packets to keep queueing delay short. The x-axis in Figure 4(a) shows the deficit saving limit $D_{max}$ and the y-axes show again the throughput and packet queueing delay of heavy and light users with TCP New Reno connections on a bottleneck link with a $D_b = 5$ ms. We observe that the throughput for heavy and light users significantly deviates for very small values of $D_{max}$ but the difference vanishes for values $D_{max} \geq 10$. We explain this phenomenon by the fact that at the beginning of a congestion phase when the buffer is filled, the heavy users quickly consume their deficit. This gives priority to packets of light users when they arrive. Also during congestion periods, light users can save deficit whenever heavy users receive deficit to send further packets although they




(a) Drop-on-enqueue for $S_B = 20$ packets.




(b) Drop-on-dequeue for $T_D = 24$ ms.

**Fig. 4.** Throughput and packet queueing delay for heavy users (HU) and light users (LU) *with LDS*. Results are shown for TCP New Reno and $D_b = 5$ ms.

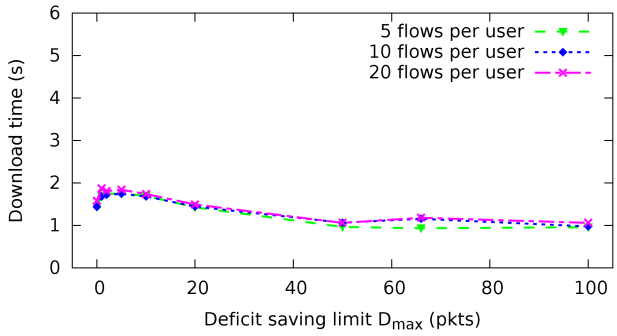
do not have packets to send. When traffic is shared about equally, the queueing delay is about 20 ms and for heavy users slightly larger than for light users.

Figure 4(b) shows similar data for drop-on-dequeue for which a delay threshold of $T_D = 24$ ms is chosen that corresponds to the transmission time of 20 packets. The difference in throughput between heavy and light users is even larger than for drop-on-enqueue. It is again decreased by an increasing value for the deficit saving limit $D_{max}$. A deficit saving limit of $D_{max} = 6$ is already enough to achieve perfect fairness. Packet queueing delay is about 10 ms and for heavy users slightly larger than for light users.

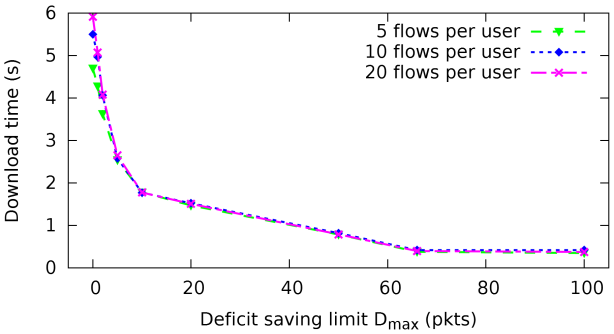
We also performed these experiments with $D_b = 50$ ms and obtained almost the same results. For TCP Cubic we received different results. In particular, with drop-on-enqueue, light users achieved significantly larger throughput than heavy users, but light users experienced also clearly more packet delay. In contrast,

drop-on-dequeue leads to a very similar queueing behavior as in Figure 4(b) for TCP New Reno.

Thus, drop-on-dequeue with DRR-LDS seems an interesting approach to maximize fairness between heavy and light users with TCP flows when low latency is required.




(a) Drop-on-enqueue for $S_B = 20$ packets.




(b) Drop-on-dequeue for $T_D = 24$ ms.

**Fig. 5.** Download time of a data chunk of 100 kB for an infrequent user in the presence of heavy background traffic. The 20 other users have 5, 10, or 20 saturated TCP connections. Results are shown for TCP New Reno and $D_b = 5$ ms.

### 4.5 Download Times with DDR-LDS

We consider the download time of an infrequent user sending small data bursts of 100 kB in the presence of a heavy load situation. The infrequent user holds a single TCP New Reno connection to download a data chunk of 100 kB for which

the duration is measured. The 20 other users hold 5, 10, and 20 connections in different experiments. The TCP connections of the 20 other users are saturated, constitute the background load, and start within the first second of the experiment. The infrequent user starts its download after 5 s. Each experiment is repeated 100 times. With perfect fairness the download time for 100 kB is 1.68 s under the assumption that TCP can initially send already sufficiently fast.

We first consider the download time for drop-on-enqueue configured with a buffer size of $S_B = 20$ packets. Figure 5(a) shows the download time for the infrequent user depending on the deficit saving limit $D_{max}$. The download time is almost independent of the background traffic. Small deficit saving limits $D_{max}$ lead to a slight increase in download time, but larger $D_{max}$ clearly decreases the download time below 1.68 s. Thus, a reduction of download time through LDS is visible but rather modest. The same results are obtained for $D_b = 50$ ms and for TCP Cubic.

To study drop-on-dequeue, we use a delay threshold of $T_D = 24$ ms. Figure 5(b) shows that without LDS ($D_{max} = 0$) the infrequent user's download time is significantly larger than for drop-on-enqueue. Larger background loads lead to longer download times without LDS, i.e., 4.8 s when the other users hold 5 connections each, 5.5 s when they hold 10 connections each, and 5.9 s when they hold 20 connections each. With very little deficit saving limit $D_{max}$, the difference in download time already vanishes. With even larger deficit saving limit $D_{max}$ the download time decreases to less than 0.4 s. This is a significant speedup even compared to the fair download time of 1.68 s. Very similar results are obtained for $D_b = 50$ ms and for TCP Cubic.

Thus, LDS in combination with DRR can expedite transactional traffic in the presence of heavy background load, with both the drop-on-enqueue or the drop-on-dequeue buffer management strategy. For drop-on-dequeue the reduction of download time is very high.

## 5 Conclusion

In this work we have shown that DRR cannot achieve equal bandwidth allocation to heavy and light users that have different numbers of saturated TCP flows, in particular if low delay is enforced through shallow buffers. A reason for this phenomenon is that empty queues do not profit in DRR's bandwidth allocation process. If low delay is enforced, even queues with active TCP flows are empty for quite some time so that it is hard for them to get their full bandwidth share. Therefore, we modified the DRR algorithm such that empty queues are respected for the bandwidth allocation process in DDR and can save deficit to a limited extent. We called this mechanism limited deficit savings (LDS) and refer to the modified DDR algorithm as DDR-LDS. Our simulation results demonstrated the throughput differences between heavy and light users and revealed that they are larger for drop-on-dequeue than for drop-on-enqueue and for TCP New Reno than for TCP Cubic. We showed that LDS significantly reduces these differences. Moreover, LDS clearly decreases download times of light users for both buffer

management strategies whereby the effect for drop-on-dequeue is larger than for drop-on-enqueue.

The study points at potential unfairness with DRR under low-delay constraints for TCP users. More research is needed to understand how DRR-LDS affects other traffic types, traffic mixes, and how the performance depends on networking parameters that typically influence TCP throughput. Moreover, measurements are needed to evaluate whether the investigated scenario is rather a corner case or whether the DRR-LDS scheduler is able to solve practical problems. Nevertheless, a variant of LDS can be identified in FQ-CoDel so that the presented mechanism has practical relevance and should be understood.

## References


1. FP7 Project (STREP): Reducing Internet Transport Latency (RITE) (Nov 2012 – Nov 2015)
2. Adams, R.: Active Queue Management: A Survey. IEEE Communications Surveys & Tutorials 15(3), 1425–1476 (2013)
3. Allman, M.: Comments on Bufferbloat. ACM SIGCOMM Computer Communication Review 43(1) (Jan 2013)
4. Bennett, J.C., Zhang, H.: $WF^2Q$ : Worst-Case Fair Weighted Fair Queueing. In: IEEE Infocom (1996)
5. Briscoe, B., Brunstrom, A., Petlund, A., Hayes, D., Ros, D., Tsang, I.J., Gjessing, S., Fairhurst, G., Griwodz, C., Welzl, M.: Reducing Internet Latency: A Survey of Techniques and their Merits. to appear in IEEE Communications Surveys & Tutorials (2016)
6. Demers, A., Keshav, S., Shenker, S.: Analysis and Simulation of a Fair Queuing Algorithm. In: ACM SIGCOMM (1989)
7. Dhamdhere, A., Dovrolis, C.: Open Issues in Router Buffer Sizing. ACM SIGCOMM Computer Communication Review 36(1), 87–92 (Jan 2006)
8. Dhamdhere, A., Jiang, H., Dovrolis, C.: Buffer Sizing for Congested Internet Links. In: IEEE Infocom. Miami, FL (Mar 2005)
9. Floyd, S., Jacobson, V.: Random Early Detection Gateways for Congestion Avoidance. IEEE/ACM Transactions on Networking 1(4), 397–413 (Aug 1993)
10. Gettys, J., Nichols, K.: Bufferbloat: Dark Buffers in the Internet. ACM Queue 9(11) (Nov 2011)
11. Gong, Y., Rossi, D., Leonardi, E.: Modeling the Interdependency of Low-Priority Congestion Control and Active Queue Management. In: International Teletraffic Congress (ITC) (2013)
12. Gong, Y., Rossi, D., Testa, C., Valenti, S., Taht, M.D.: Fighting the Bufferbloat: on the Coexistence of AQM and Low Priority Congestion Control. In: IEEE INFOCOM Workshop on Traffic Measurement and Analysis (2013)
13. Hoeiland-Joergensen, T., McKenney, P., Taht, D., Gettys, J., Dumazet, E.: RFC8290: The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm. https://tools.ietf.org/html/rfc8290 (Jan 2018)
14. Hohlfeld, O., Pujol, E., Ciucu, F., Feldmann, A., Barford, P.: BufferBloat: How Relevant? A QoE Perspective on Buffer Sizing. Tech. Rep. 2012-11, TU Berlin, Faculty of Electrical Engineering and Computer Science (Nov 2012)

15. IETF Working Group on Active Queue Management and Packet Scheduling (AQM): Description of the Working Group. http://tools.ietf.org/wg/aqm/charters (2013)
16. IETF Working Group on Congestion Exposure (CONEX): Description of the Working Group. http://tools.ietf.org/wg/conex/charters (2010)
17. IETF Working Group on RTP Media Congestion Avoidance Techniques (RMCAT): Description of the Working Group. http://tools.ietf.org/wg/rmcat/charters (2012)
18. Internet Society: Bandwidth Management – Internet Society Technology Roundtable Series. http://www.internetsociety.org/sites/default/files/BWroundtable_report-1.0.pdf (Nov 2012)
19. Internet Society: Report on the Workshop on Reducing Internet Latency. http://www.internetsociety.org/latency2013 (Dec 2013)
20. Jiang, H., Liu, Z., Wang, Y., Lee, K., Rhee, I.: Understanding Bufferbloat in Cellular Networks. In: Workshop on Cellular Networks: Operations, Challenges, and Future Design (CellNet) (Aug 2012)
21. Khademi, N., Ros, D., Welzl, M.: The New AQM Kids on the Block: Much Ado About Nothing? Tech. Rep. Technical Report 434, University of Oslo, Dept. of Informatics (2013)
22. McKenney, P.E.: Stochastic Fairness Queueing. In: IEEE Infocom (1990)
23. Nagle, J.: RFC970: On Packet Switches with Infinite Storage (Dec 1985)
24. Nichols, K., Jacobson, V., McGregor, Ed., A., Iyengar, Ed., J.: RFC8289: Controlled Delay Active Queue Management. https://tools.ietf.org/html/rfc8289 (Jan 2018)
25. Nichols, K., Jacobson, V.: Controlling Queue Delay. ACM Queue 10(5) (May 2012)
26. Pan, R., Natarajan, P., Piglione, C., Prabhu, M.S., Subramanian, V., Baker, F., VerSteeg, B.: PIE: A Lightweight Control Scheme to Address the Bufferbloat Problem. In: IEEE Workshop on High Performance Switching and Routing (HPSR) (2013)
27. Ramakrishnan, K., Floyd, S., Black, D.: RFC3168: The Addition of Explicit Congestion Notification (ECN) to IP (Sep 2001)
28. Shreedhar, M., Varghese, G.: Efficient Fair Queuing Using Deficit Round Robin. IEEE/ACM Transactions on Networking 4(3), 375 – 385 (Jun 1996)
29. Suter, B., Lakshman, T.V., Stiliadis, D., Choudhury, A.K.: Buffer Management Schemes for Supporting TCP in Gigabit Routers with Per-flow Queueing. IEEE Journal on Selected Areas in Communications 17(6), 1159 – 1169 (Jun 1999)
30. Varga, A.: INET-2.2 released. http://inet.omnetpp.org/ (Aug 2013)
31. Varga, A.: OMNeT++ 4.3.1 released. http://www.omnetpp.org/ (Sep 2013)
32. Wand, S.: Network Simulation Cradle. http://research.wand.net.nz/software/nsc.php (2012)
33. White, G., Pan, R.: RFC8034: Active Queue Management (AQM) Based on Proportional Integral Controller Enhanced (PIE) for Data-Over-Cable Service Interface Specifications (DOCSIS) Cable Modems. https://tools.ietf.org/html/rfc8034 (Feb 2017)
34. White, G.: Data-over-Cable Service Interface Specification – MAC and Upper Layer Protocols Interface Specification. Tech. Rep. CM-SP-MULPIv3.1-I01-131029, Cable Television Laboratories, Inc. (Oct 2013)
35. White, G.: Active Queue Management in Docsis 3.X Cable Modems. Tech. rep., Cable Television Laboratories, Inc. (May 2014)
36. Zhang, L.: VirtualClock: A New Traffic Control Algorithm for Packet Switching Networks. In: ACM SIGCOMM (1990)