Using Data Center TCP (DCTCP) in the Internet

Mirja Kühlewind*, David P. Wagner[†], Juan Manuel Reyes Espinosa[†], Bob Briscoe[‡]
*Communication Systems Group, ETH Zurich, Switzerland
mirja.kuehlewind@tik.ee.ethz.ch
[†]Institute of Communication Networks and Computer Engineering, University of Stuttgart, Germany
david.wagner@ikr.uni-stuttgart.de
[‡]BT Research, Ipswich, UK
bob.briscoe@bt.com

Abstract

Data Center TCP (DCTCP) is an Explicit Congestion Notification (ECN)-based congestion control and Active Queue Management (AQM) scheme. It has provoked widespread interest because it keeps queuing delay and delay variance very low. There is no theoretical reason why Data Center TCP (DCTCP) cannot scale to the size of the Internet, resulting in greater absolute reductions in delay than achieved in data centres. However, no way has yet been found for DCTCP traffic to coexist with conventional TCP without being starved. This paper introduces a way to deploy DCTCP incrementally on the public Internet that could solve this coexistence problem. Using the widely deployed Weighted Random Early Detection (WRED) scheme, we configure a second AQM that is applied solely to ECN-capable packets. We focus solely on long-running flows, not because they are realistic, but as the critical gating test for whether starvation can occur. For the non-ECN traffic we use TCP New Reno; again not to seek realism, but to check for safety against the prevalent reference. We report the promising result that, not only does the proposed AQM always avoid starvation, but it can also achieve equal rates. We even derived how the sharing ratio between DCTCP and conventional TCP traffic depends on the various AQM parameters. The next step beyond this gating test will be to quantify the reduction in queuing delay and variance in dynamic scenarios. This will support the standardization process needed to define new ECN semantics for DCTCP deployment that the authors have started at the IETF.

I. Introduction

Data Center TCP (DCTCP) [1] has provoked widespread interest because it keeps queuing delay and delay variance very low relative to the propagation delay across the data centre. Alizadeh [2] shows that the DCTCP approach can scale to networks with much larger propagation delay. Queuing delay and delay variance grow proportionately in absolute terms. Nonetheless they remain low relative to the propagation delay. However, DCTCP requires changes to all three main parts of the system: network buffers, receivers and senders. Therefore DCTCP deployments have been confined to environments like private data centers where a single administration can upgrade all the parts at once.

The reason for the need to change all three parts is a different interpretation of the network's congestion signals than the usual one. The difference concerns both the amount of congestion signaled in one round of feedback and the congestion's extent in time: DCTCP's design is based on signaling congestion immediately when a rather small queue builds up, leaving smoothing to the sender. In contrast, today's Internet is based on smoothing in the network and then signaling only when a severe queue has already built up. Based on the interest in DCTCP as well as other congestion management system, current standardization activity to change the semantics of Explicit Congestion Notification (ECN) is under way in the Internet Engineering Task Force (IETF) [3], [4]. This change to congestion semantics allows DCTCP to provide low latency even at large buffers, if all involved players respect these new semantics. This is feasible given there has been no effective ECN deployment on the public Internet.

While it is straightforward to coordinate senders and receivers [4] even in mixed environments such as the Internet, network nodes need to cope with heterogeneous traffic. This paper introduces a way to configure existing switches and routers that allows DCTCP and non-DCTCP to share a queue. The Active Queue Management (AQM) of this queue must support both congestion semantics, the fast and fine-grained feedback needed by DCTCP and the more coarse-grained and slower signal needed by conventional Transmission Control Protocol (TCP) traffic. Since both types of traffic fill the shared queue, low delay and high throughput can only be achieved, if the traffic is dominated by DCTCP. We aimed at finding configurations, that allow an evolution from today's 100% conventional TCP traffic to DCTCP dominated traffic. Network operators may initially configure the dual AQM close to the ideal for loss-based congestion control, but they can shift closer to the ideal of low delay as the proportion of DCTCP traffic grows.

So we target three main questions: First, how do Random Early Detection (RED)-based AQM configurations influence the sharing between Reno and DCTCP flows; can starvation always be avoided; and can at least roughly equal sharing be achieved? Second, for which Internet scenarios and configurations do DCTCP users benefit in terms of low delay? And third: which utilization can be achieved for the promising configurations?

We evaluated these questions by simulations integrating patched Linux kernels under constant conditions in simple scenarios. Next, since this initial 'gating test' shows promise, we plan to evaluate the idea in a wide range of more realistic scenarios. In this paper we show the general feasibility of our approach and give hints for how to evolve configuration control assuming the benefits lead to an increasing proportion of DCTCP flows in the future Internet.

We give an overview on DCTCP in Section II. In Section III we present modifications to algorithm and implementation of DCTCP and introduce our proposed dual AQM scheme. In Section IV we present our results and show that a stable operation with lower latency including equal sharing, if desired, is possible. We also derived rules for defining AQM parameters in order to achieve both, lower latency and equal sharing.

II. Overview of DCTCP

DCTCP is a combination of a congestion control scheme and AQM scheme that is based on new semantics of congestion notification using ECN signaling. DCTCP implements three changes: a different reaction to congestion in the sender, a specific RED configuration in the network nodes, and a more accurate congestion feedback mechanism from the receiver to the sender.

A. Simple Marking Scheme

The AQM scheme for DCTCP operation is deceptively simple: if the current instant queue occupancy is larger than a certain threshold K, every arriving packet will be marked. This mechanism can be implemented as a specific parameterization of RED [5]. RED probabilistically decides about the marking of arriving packets based on the average queue occupancy, calculated as a weighted moving average with the weighting factor w. Only above a minimum threshold (Min_Thresh) arriving packets will be marked with linearly increasing probability, as also displayed in Figure 3, reaching up to a maximum marking probability (Max_Prob) at the maximum threshold (Max_Thresh) and a probability of 1 above. The proposed AQM scheme for DCTCP can be realized by using RED with $Min\_Thresh = Max\_Thresh = K$ and $w = 1$.

B. Congestion Control Algorithm

A TCP sender maintains a congestion window ($cwnd$), giving the allowed number of packets in flight during one Round Trip Time (RTT). With DCTCP, when an ECN-Echo arrives, $cwnd$ is updated to reflect not just the existence of some congestion within a round trip, but the exact proportion of congestion marks. This is achieved according to the following equation

$$cwnd \leftarrow (1 - \alpha/2) * cwnd \tag{1}$$

where α is the moving average of the fraction of marked packets in the last RTT. Its calculation is given by

$$\alpha \leftarrow (1 - g) * \alpha + g * F \tag{2}$$

where F is the fraction of the marked packets in the last RTT and g is a weighting factor. g is recommendationed to be set to $1/2^4$ in [1]. α is updated once per RTT. This congestion control algorithm allows the sender to gently reduce its congestion windows in case of low fraction of markings, whereas strong reductions are performed in case of a high degree of congestion.

C. Enhanced ECN Feedback

ECN allows network nodes to notify of congestion by setting a flag in the Internet Protocol (IP) header (Congestion Experienced (CE) codepoint), with no need to drop packets. A host receiving a CE-marked packet will send ECN-Echoes (ECE) in every TCP acknowledgement until it receives a Congestion Window Reduced (CWR)-flagged TCP packet. By this mechanism only one congestion feedback can be sent per RTT which is appropriate for conventional TCP congestion control but not for DCTCP. Thus DCTCP changes the ECN feedback mechanism. It aims to get exactly one ECN-Echo for each CE-marked packet. However, to be able to use delayed acknowledgements, Alizadeh et al. define in [1] a two state machine for handling ECN feedback. Note that there is no negotiation as DCTCP assumes that the receiver is DCTCP-enabled. For wider use in the Internet, the authors are standardizing a negotiation phase [3].

III. Modifications

Our evaluation is based on a Linux patch provided by the University of Stanford [6] applied to the Linux kernel version 3.2.18. In the initial phase of our investigations we observed unexpected and undesired behavior of that implementation which we fixed by minor modifications described in the following section. Furthermore, we implemented two algorithmic modifications to provide a faster adaptation to the current congestion level. Finally, we present our dual AQM scheme in Subsection III-C.

A. Implementation in Linux

a) Finer resolution for α value: In the provided implementation [6] the resolution of α was limited to minimum value of $1/2^{10}$. Because of this, in our simulation scenario the congestion window converged to a fixed value in a situation with very few ECN markings. For our investigations we changed the minimum resolution to $1/2^{20}$. It should be noted that for large congestion windows and very low marking rates, an even finer resolution might be necessary.

Setting of the Slow Start threshold: In the provided DCTCP implementation the Slow Start threshold ($ssthresh$) is incorrectly set to the current $cwnd$ value after a reduction. In our implementation we correctly reset the $ssthresh$ to the $cwnd - 1$ instead. With the original patch a DCTCP sender was in Slow Start ($cwnd <= ssthresh$) after each decrease and thus immediately increases (by one packet) on arrival of the first ACK, then leaving Slow Start and correctly entering Congestion Avoidance. As with DCTCP not every window recalculation causes a window reduction, therefore this error caused a non-linear increase in a noticeable range.

b) Allow the congestion window to grow in CWR state: While the Linux congestion control implementation in general does not allow any further window increases during roughly one RTT after the reception of a congestion signal, this does not seem to be appropriate for DCTCP. Thus in our implementation we allow the congestion window to grow even during this so-called CWR state. Moreover, if no reduction was performed, we do not reset snd_cwnd_cnt, which maintains when to increase the window next, to preserve the linear increase behavior.

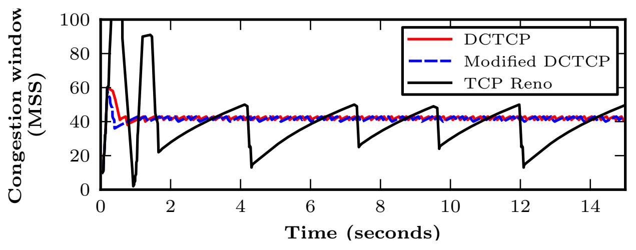

Fig. 1. Congestion window of single flows

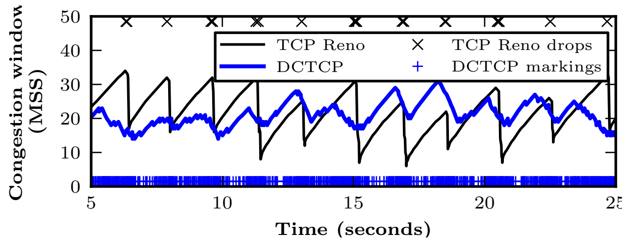

Fig. 2. Congestion window and mark/drop events for one TCP Reno and one DCTCP host sharing an accordingly configured queue

B. Algorithmic Modifications

c) Continuous update of α: As mentioned in Section II-B, α is updated only once per RTT. With such a periodic update scheme, α might not catch the maximum congestion level and, even worse, might still reflect an old value when the congestion window reduction is performed. To avoid this, we update α on the reception of each acknowledgement. It must be mentioned that for with the modificaion also the weighting factor g must be chosen differently because α is recalculated more often. Therfore we set g to be $1/2^8$ instead of $1/2^4$ to compensate for this effect, thus making the behavior similar to the original DCTCP patch in our rather static evaluation scenarios. However, the right choice of g depends on the absolute number of markings, and thus number of recalculations performed, and therefore actually depends on the current number of packets in flight. This dependency could be compensated by normalizing the fraction of marked packets F with the current number of packets in flight, or simply the current congestion window value.

d) Progressive congestion window reduction: In the original implementation the congestion window is recalculated as soon as the CWR state is entered. But, as explained above, the actual congestion level would need to be determined over the following RTT in which further congestion signal are expected to be received. We cannot wait one whole RTT to perform any window reductions, as this would cause further unnecessary congestion. Thus we decrease the congestion window progressively on reception of each ECN-Echo. For each recalculation we use the congestion window value $cwnd\_max$ from the start of the CWR state and reset the congestion window only if the resulting value is lower than the current value.

Figure 1 shows the congestion window of one DCTCP flow either using the original patch or our modification in comparison to one TCP Reno flow. It can be seen that after the Slow Start phase our implementation adapts faster but otherwise the behavior is similar, as desired.

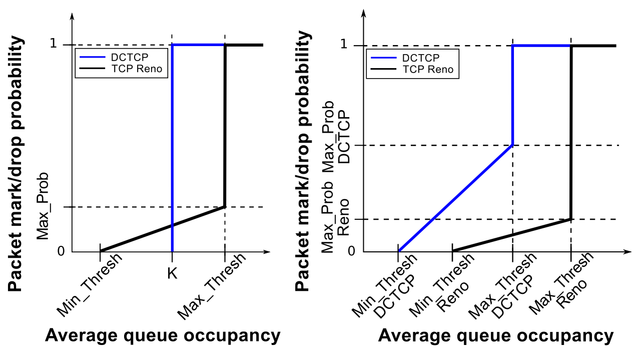

Fig. 3. Packet mark probability calculation

C. Dual AQM Scheme

The packets of DCTCP and other TCP flows need to be handled differently in the AQM scheme of the bottleneck network node according to the different congestion signal semantics. We propose an AQM scheme based on one shared queue but applying two differently parameterized instances of the RED algorithm, one for non-ECN traffic and one for ECN traffic. Our scheme classifies the traffic based on the ECN-capability, thus packets will respectively be marked or dropped to notify of congestion. This approach would probably result in low throughput for ECN-enabled end-systems that still use conventional congestion control such as Reno. However, given this ECN standard was defined in 2001 and has hardly seen any active use, it is unlikely this is an important factor.

Instead we propose to standardize an ECN signal that signals congestion immediately, allowing the end hosts to distinguish between smoothed and immediate congestion notification. DCTCP together with more accurate ECN feedback, as already under standardization [3], [4], could be re-implemented and turned on after ECN capability negotiation with the server. The much greater performance benefits of DCTCP, could then incentivize OS developers to deploy DCTCP with ECN turned on by default.

Figure 2 shows exemplarily the resulting congestion signals along with the congestion window of one Reno and one DCTCP flow equally sharing the bandwidth. The used parameter set is derived from our investigations described later on in the second part of Section IV-C.

IV. Preliminary Evaluation

In this evaluation we investigated DCTCP with the proposed dual AQM scheme in a simplified scenario to show feasibility. Our parameter study shows that a large range of configuration can be used to achieve different operation points in link utilization, queue occupancy (and thus latency) and bandwidth sharing between multiple flows. We investigated the two approaches of RED parameterization for the DCTCP traffic as illustrated in Figure 3: i) (left) a degenerate configuration with $Min\_Thresh\_DCTCP = Max\_Thresh\_DCTCP = K$ creating a simple marking threshold as originally proposed for DCTCP or ii) (right) with $Min\_Thresh\_DCTCP < Max\_Thresh\_DCTCP$ as in standard RED configurations as described in III-C, i.e. either using a marking threshold K or a marking slope. The selected parameterization covers only

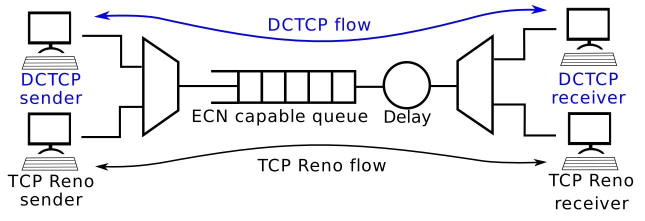

Fig. 4. Simulation scenario (only forward direction)

a limited range of the large parameter set but presents the two most interesting cases. Other scenarios need to be investigated before applying our approach to the Internet to cover corner cases.

A. Simulation Environment and Scenario

We evaluated our approach based on simulations using the IKR SimLib [7], an event-driven simulation library with an extension to integrate virtual machines [8] running a Linux kernel with our modified DCTCP implementation.

As shown in Figure 4, the simulation scenario consists of four hosts, two senders and two receivers, connected by a shared link of 10 Mbps with a single bottleneck queue plus the corresponding return path and an RTT of 25 ms, resulting in a Bandwidth Delay Product (BDP) of 31250 bytes. One pair of hosts uses DCTCP with ECN, while the others use TCP Reno without ECN support. Each sender has data to send at any time and uses one or more long-lived connections. The queue implements the dual AQM scheme described in Section III-C.

To limit the parameter space, we fixed most of the RED parameters for non-ECN traffic to the recommended values in [9]; using a weighting factor w_Reno of 0.002 and setting Max_Prob_Reno to 0.1. Moreover, we configured the maximum threshold to three times the minimum threshold: $Max\_Thresh\_Reno = 3 * Min\_Thresh\_Reno$. Min_Thresh_Reno is the parameter we vary because it determines the queuing-induced latency (if non-DCTCP traffic exists).

B. Using a Marking Threshold

For this approach both DCTCP thresholds are set to the same value K and smoothing is turned off by setting $w\_DCTCP = 1$, as was originally proposed for DCTCP. We vary the step threshold K in relation to Min_Thresh_Reno for several values of Min_Thresh_Reno smaller or equal than the BDP. Note that the buffer size needed by one Reno flow to fully utilize the link is one BDP. K must be between the minimum and maximum threshold of the Reno traffic to avoid that only one flow gets almost all capacity, thus K/Min_Thresh_Reno is varied between 1 and 3. We aim at equal shares of DCTCP and non-DCTCP at high utilization and lower delay, so in Figure 5 we plot the ratio between Reno and DCTCP traffic, the utilization of the bottleneck link and the average queue occupancy. The steps that can be seen for the smaller thresholds, most prominently for $Min\_Thresh\_Reno = BDP/8$, are a consequence of the fixed Maximum Transmission Unit (MTU) and the small marking threshold: if all packets are 1500 Bytes, it makes no difference if the marking threshold is $7812B = 2 * BDP/8$ or $8203B = 2.1 * BDP/8$, both are able to hold 5 packets without marking.

TABLE I. (M)IN_THRESH_RENO (IN BDPS), K/M AT MAXIMUM FAIRNESS, LINK (U)TILIZATION AND QUEUE (O)CCUPANCY OF (D)CTCP AND (R)ENO

M	K/M	U_{DR}	O_{DR}	U_{2R}	O_{2R}
$1/8$	2.3	0.98	0.207	0.946	0.216
$1/4$	1.73	0.987	0.335	0.968	0.336
$1/2$	1.34	0.996	0.548	0.986	0.524
$1/\sqrt{2}$	1.26	0.999	0.724	0.995	0.679
1	1.16	1.000	0.99	0.999	0.914

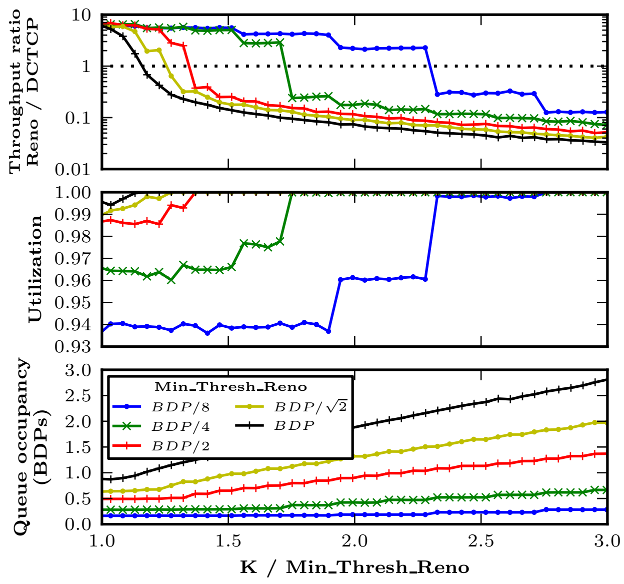

Fig. 5. Results when using marking threshold

As it can be seen in the throughput plot of Figure 5, for any Min_Thresh_Reno there is a K that results in equal bandwidth sharing (dotted line) between the DCTCP and the Reno flow. Table IV-A lists these values and the respective utilization and queue occupancy for either one DCTCP and one Reno flow competing or, for comparison, two Reno flows only. Especially, when the minimum threshold is chosen very low to $1/8$*BDP, it can be seen that DCTCP increases the utilization (up to 4 %) while the average queue remains about the same.

These results show that with this configuration scheme a lower delay can be traded for a only slightly lower utilization while sharing equally with Reno flows. More specifically, the queue occupancy can be reduced by a factor of four while maintaining equal bandwidth sharing and high (98 %) utilization.

Several DCTCP flows: We also investigated the effect of increased proportion of DCTCP traffic on the utilization for a Min_Thresh_Reno as low as $BDP/8$. For that purpose, we ran experiments where one Reno flow competes with N DCTCP flows. In Figure 6 the throughput ratio between Reno and the average of the N DCTCP flows is shown for $N = 1...5$, along with the corresponding utilization and queue occupancy. As expected the utilization increases with the number of DCTCP flows. Equal sharing can only be achieved for

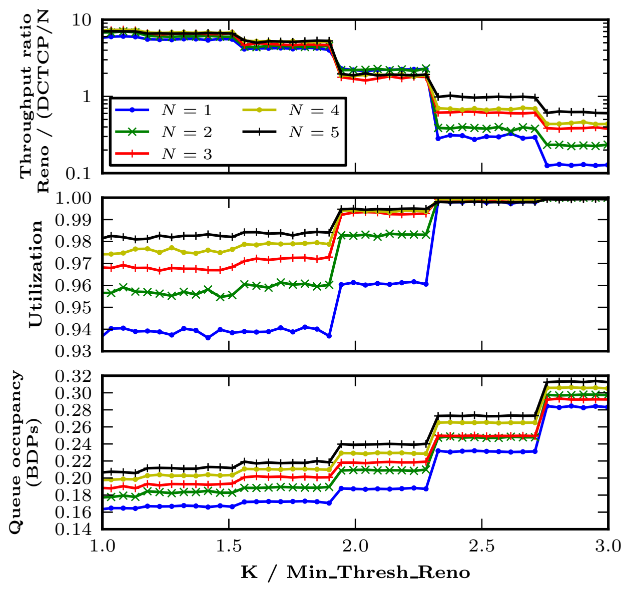

Fig. 6. Results with multiple DCTCP flows for $Min\_Thresh\_Reno = 1/8 * BDP$

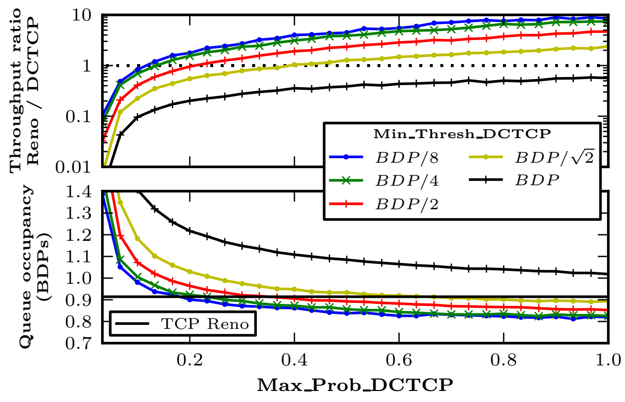

Fig. 7. Results when using marking slope

settings with large values of K/Min_Thresh_Reno which also increases the average queue occupancy. Thus with a larger proportion of DCTCP in a (future) traffic mix, the AQM thresholds could be lowered while maintaining high utilization.

C. Using a Marking Slope

The alternative parameterization for the DCTCP traffic uses a conventional RED configuration forming a slope of increasing marking probability depending on the average queue occupancy (in contrast to using a step function of instantaneous queue length). We also studied the influence of the weighting factor w_DCTCP. As expected for a non-dynamic scenario with just long-running flows, we found that it has only minor influence on bandwidth sharing and thus chose the same value for w_DCTCP as for w_Reno of 0.002. For this study we fixed Min_Thresh_Reno to BDP, resulting in $Max\_Thresh\_Reno = 3 * BDP$. We investigate values for Min_Thresh_DCTCP smaller than BDP and shifted Max_Thresh_DCTCP to $Min\_Thresh\_DCTCP + 2 * BDP$ to keep the same distance between the thresholds and thus the same slope which again is a simplification to narrow

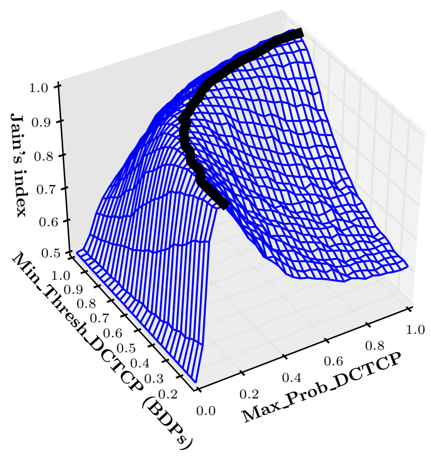

Fig. 8. Jain's fairness index

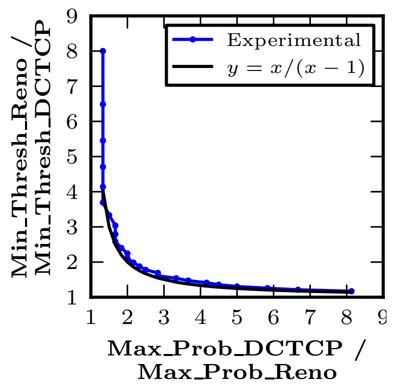

Fig. 9. Maximum fairness

our parameter set. Figure 7 shows the throughput ratio between Reno and DCTCP and the queue occupancy when varying Max_prob_DCTCP from 0 to 1. The plot shows equal sharing is possible for many parameterizations, except for the $Min\_Thresh\_DCTCP = BDP$. This is expected since both flows get the same feedback rate but Reno reacts by halving the sending rate while DCTCP usually will decrease less (depending on the number of markings). Figure 8 shows a 3-dimensional plot of the Jain's fairness index [10] depending on Min_Thresh_DCTCP and Max_Prob_DCTCP on the left. The highlighted ridge marks a fairness index of one. Figure 9 shows the parameter combinations of maximum fairness. The function $y = x/(x-1)$ is overlaid to illustrate that it fits quite closely to the measured maximum fairness configurations. These results suggest that we can achieve equal sharing for a parameterization according to the following rule:

$$\frac{Min\_Thresh\_DCTCP}{Min\_Thresh\_Reno} = \frac{\frac{Max\_Prob\_DCTCP}{Max\_Prob\_Reno} - 1}{\frac{Max\_Prob\_DCTCP}{Max\_Prob\_Reno}} \quad (3)$$

That means for a given configuration for Reno, there is just one parameter to choose, Min_Thresh_DCTCP or Max_Prob_DCTCP.

Equal Sharing Configurations: Since this formula provides configurations that implement (about) equal sharing, we scale the maximum marking probability by Min_Thresh_DCTCP and altered only the minimum

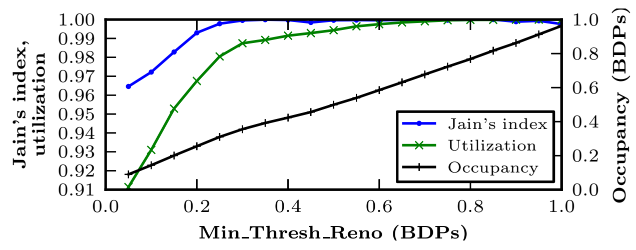

Fig. 10. Equal Sharing

threshold for Reno traffic in this evaluation. That means we set Max_Prob_Reno to $0.1/(Min\_Thresh\_Reno/BDP)$ and Max_Prob_DCTCP to $0.2/(Min\_Thresh\_Reno/BDP)$. We show results for $Min\_Thresh\_DCTCP = 1/2 * Min\_Thresh\_Reno$.

Figure 10 shows Jain's fairness index, utilization and queue occupancy. As it can be seen, such configurations achieve almost maximum fairness in terms of equal sharing and the queue occupancy depends about linearly on Min_Thresh_Reno. The achieved utilization is close to 100 % with a Min_Thresh_Reno of only 0.3*BDP. For the very simple scenario considered, this finding allows to define a trade off between delay and utilization while keeping the sharing equal thus being "TCP-friendly".

V. Conclusions

In this paper we propose a new dual AQM scheme that can be implemented based on Weighted RED (WRED) to incrementally deploy DCTCP with its different congestion semantics in the public Internet. Today ECN sees only minimal deployment, but activities in standardization are under way to re-define the congestion feedback mechanism and its meanings. We argue that a classification solely based on the ECN-capability of the traffic provides an opportunity for actual DCTCP deployment in the Internet. Therefore, we evaluated the possibility of concurrent usage of DCTCP with other conventional TCP congestion control. We evaluated two RED configurations for providing ECN-based feedback for DCTCP traffic: i) a marking threshold K as proposed by the original DCTCP approach and ii) a marking slope as in standard RED configurations. We showed that both approaches can be configured for stable operation, where the proportions of DCTCP and Reno traffic converge to a certain ratio or even to an equal rate, if desired. Moreover, we found a formula for RED parameters that always results in equal sharing between DCTCP and non-DCTCP. This relation allows high utilization to be traded off against low delay. We showed that, even with the minimum threshold set very low to maintain low latency, utilization increases with a larger fraction of DCTCP traffic.

This study is only a first step to show that the way proposed to deploy DCTCP in the Internet would at least give a reasonable share of capacity to long-running flows; while still reducing latency and maintaining high utilization. Further evaluation is needed using scenarios with all kinds of traffic models, e.g. with more and not only long-running flows and different shares of DCTCP and conventional TCP flows. Our interest lies also in a wider parameter study focusing on scenarios with ECN marking based on the instantaneous queue length only, as DCTCP already implements smoothing itself. We expect further advantages from DCTCP's reaction to congestion when flows with very small and very large RTTs are sharing the same bottleneck. We also need to show that endpoints and network nodes with the new semantics can safely coexist with any legacy ECN endpoints or network nodes, in case they are deployed without update.

The proposed way to deploy DCTCP in the Internet requires instantaneous and more accurate ECN feedback. Today ECN is defined as a "drop equivalent" and therefore provides only small performance gains and consequently has not seen wide deployment. With a change in semantics, ECN could be used as an enabler for new low latency services also implementing a different response to congestion, similar to DCTCP.

Apart from a more accurate ECN signal, where a proposal by the authors has already been adopted onto the IETF's agenda, we also see a need to standardize a change to the semantics of ECN to provide immediate congestion information without any further delays in the network. This work provides further input on the needs for a future, immediate, and therefore more beneficial ECN-based congestion control loop and proposes an approach for how congestion control could react to such signal.

VI. Acknowledgments

This work was performed while the first author was still with IKR, University of Stuttgart, Germany This work is part-funded by the European Community under its Seventh Framework Programme through the ETICS project and the Reducing Internet Transport Latency (RITE) project (ICT-317700). The views expressed here are solely those of the authors.

References

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "DCTCP: Efficient packet transport for the commoditized data center," in *Microsoft Research publications*, January 2010.

[2] M. Alizadeh, A. Javanmard, and B. Prabhakar, "Analysis of DCTCP: stability, convergence, and fairness." in *SIGMETRICS*. ACM, 2011.

[3] B. Briscoe, R. Scheffenegger, and M. Kuehlewind, "More Accurate ECN Feedback in TCP: draft-kuehlewind-tcpm-accurate-ecn-03," IETF, Internet-Draft, Jul. 2014, (Work in Progress).

[4] M. Kühlewind, R. Scheffeneger, and B. Briscoe, "Problem Statement and Requirements for a More Accurate ECN Feedback: draft-ietf-tcpm-accecn-reqs-06," IETF, Internet-Draft, Jul. 2014, (Work in Progress).

[5] S. Floyd and V. Jacobson, "Random Early Detection gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, pp. 397–413, Aug. 1993.

[6] "DCTCP patch for linux 3.2," https://github.com/mininet/mininet-tests/blob/master/dctcp/0001-Updated-DCTCP-patch-for-3.2-kernels.patch, 2014.

[7] "IKR Simulation Library," http://www.ikr.uni-stuttgart.de/Content/IKRSimLib/, 2014.

[8] T. Werthmann, M. Kaschub, M. Kühlewind, S. Scholz, and D. Wagner, "VMSimInt: A Network Simulation Tool Supporting Integration of Arbitrary Kernels and Applications," in *Proceedings of the 7th ICST Conference on Simulation Tools and Techniques (SIMUTools)*, 2014.

[9] S. Floyd, "RED: Discussions of setting parameters," http://www.icir.org/floyd/REDparameters.txt, November 1997.

[10] R. Jain, D. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," *DEC Research Report TR-301*, September 1984.