Evaluating CAIA Delay Gradient as a Candidate for Deadline-Aware Less-than-Best-Effort Transport

Tor Christian Tangenes, David A. Hayes, Andreas Petlund and David Ros

Simula Research Laboratory
P.O. Box 134, 1325 Lysaker, Norway
Email: tctangen@student.matnat.uio.no, {davidh,apetlund,dros}@simula.no

***Abstract*—Less-than-best-effort (LBE) congestion control offers a low-priority service for applications tolerant to high latency and low throughput, like peer-to-peer file transfers or automatic software updates. There are, however, situations where it would be beneficial for the application to specify a soft deadline for task completion. Examples of such situations could be completion of backup tasks or synchronisation between CDN data centres. Since network conditions change over time, a deadline-aware LBE (DA-LBE) congestion control would need the ability to dynamically adapt how aggressively it competes for capacity to meet the soft deadline, trading low-priority behaviour for timeliness. One candidate that shows promise as a LBE congestion control is CAIA Delay Gradient (CDG). CDG uses changes in measured end-to-end delay to control the congestion window. CDG has several parameters that might help tune its "aggressiveness" in a way that might help achieve the goal of DA-LBE congestion control. We have evaluated CDG in order to establish how it can be tuned to exhibit different degrees of LBE behaviour under varying network conditions. Our results show that it is possible to control CDG to vary its aggressiveness in a consistent way, making it a prime candidate to implement a DA-LBE congestion control system.**

***Index Terms*—Congestion control; less-than-best-effort service; deadline-aware congestion control; scavenger service.**

I. Introduction

Commonly used TCP congestion control (CC) variants, like CUBIC or Reno, provide a best-effort (BE) end-to-end service. When a bottleneck is congested, congestion control tries to ensure that all the flows in the bottleneck get a fair share of the available capacity. There are, however, applications that do not have strong requirements for low latency or persistently high throughput, like software updates, background downloads or backups. Ideally, traffic from such applications should yield when other traffic arrives and quickly grab available resources when the bottleneck is not congested. A *less-than-best-effort* (LBE) or *scavenger* congestion control can provide such a service.

Applications like synchronisation between CDN data centres or distributed backups could be well served by an LBE transport, however, these applications may have soft timeliness constraints; ideally, data transfers should be finished before a deadline, but not meeting the deadline would not break the application. A *deadline-aware* less-than-best-effort (DA-LBE) service would enable timely completion of transfers if the capacity allows it, while still yielding to regular BE traffic as much as possible. In order for DA-LBE to work, the lack of aggressiveness in transmission, or "LBE-ness" as we will call it henceforth, needs to be automatically adjustable in such a way that (a) it can compete on up to near-equal terms with a regular BE flow when needed to comply with a soft deadline, or (b) consume very little resources when there is plenty of time to reach the deadline. To dynamically adapt its sending rate, a DA-LBE transport needs to consider the time remaining until the deadline and the amount of data to be sent yet, balancing LBE-ness and timeliness.

A DA-LBE congestion control could be designed from scratch, but it should also be possible to leverage an existing, non-deadline-aware method. The final goal of our work is to explore the latter option, that is, whether it is possible to introduce deadline-awareness into an available LBE algorithm, simply by dynamically tuning the sender's parameters. This would allow the introduction of a new transport service—i.e., LBE transport with soft deadlines—while avoiding major protocol changes—e.g., new packet formats, or having to change both sender and receiver—, easing deployment.

In order to find candidates for evaluation as a DA-LBE mechanism, we have formulated three requirements candidate congestion control algorithms should satisfy:

1) *LBE-ness:* The DA-LBE candidate must have the properties of an LBE congestion control: the ability to quickly yield when BE traffic arrives and the ability to quickly grab capacity when it becomes available. For fairness reasons, a DA-LBE flow should not seize more capacity than a BE flow under similar conditions.
2) *Tunability:* The presence of sender parameters that can be tuned in a consistent way under different network conditions, so as to change the LBE-ness of the congestion control mechanism predictably.
3) *Operating System (OS) availability:* The mechanism should be present in a stock OS for experimentation and deployment.

Several LBE congestion control mechanisms have been designed to carry background traffic (see [1] for a survey), however, most have not yet seen any significant deployment or real-world implementation. The majority of LBE CC methods use measured *delays*, rather than packet loss (or ECN marks),

ISBN 978-3-901882-94-4

as indication of congestion. The intuition behind this is that increasing queuing delay indicates congestion *buildup* faster than loss, allowing for a faster reaction to congestion and yielding earlier to other competing flows.

Today, the most common LBE transport protocol is the Low Extra Delay Background Transport (LEDBAT) protocol [2], which is used by MacOS X for software updates and several common BitTorrent clients [3]. LEDBAT is a delay-based CC that aims at utilising spare capacity, while limiting to a target value the extra queuing delay along its forward path. LEDBAT makes use of a measured "base delay" that is meant to be the one-way delay when the bottleneck queue is empty. Some studies have found that LEDBAT may present issues like intra-protocol unfairness and non-LBE behaviour (see e.g. [4]–[6]), hence, we will not consider it further.

Other potential LBE candidates are Fuzzy Lower-than-best-effort Transport Protocol (FLOWER) [6] and the well-known TCP Vegas algorithm [7]. FLOWER solves some of LEDBAT's problems by replacing the proportional window control scheme with a fuzzy logic controller. However, FLOWER is fairly new, and is to our knowledge not yet part of any operating system's stock TCP implementation. Vegas does show LBE behaviour (even if it was not designed as such) and has been implemented in OS kernels, however, its dependence on round-trip delays (RTTs) to adjust its sending rate make it susceptible to react to delay fluctuations in the reverse path, and is not immune to inaccuracies in base delay estimation [1].

We have therefore focused our efforts on a delay-based CC algorithm named CAIA[1] Delay-gradient (CDG) [8], originally designed to coexist with traditional loss-based CC. CDG does not require the base delay, like Vegas or LEDBAT. Instead, the CDG algorithm adjusts the congestion window (cwnd) by comparing the *relative change in delay* (i.e., delay gradient) between RTT rounds and backs-off probabilistically according to the value of the gradients. An adjustable parameter can be used to scale the probability of backing off. As we will show later, we can use this scaling parameter to tune CDG's degree of LBE behaviour towards loss-based CC. CDG is available in both the Linux and FreeBSD kernels with several tunable parameters available to the application programmer, including the scaling parameter. We have thus chosen CDG for further evaluation in the search of a good DA-LBE candidate, and this evaluation is the topic of this paper.

The rest of the paper is organised as follows. In Section II we present the CDG algorithm in detail, and how it could be dynamically tuned to control its aggressiveness. Section III describes our experimental environment for evaluation of CDG and discusses the results. Finally, Section IV summarizes and concludes the paper.

II. CAIA Delay-gradient (CDG)

CAIA Delay-gradient (CDG) [8] is a delay based congestion control mechanism that reacts to perceived congestion in a probabilistic manner. The overall probability of backing-off to congestion can be adjusted by using a scaling parameter, G, which is configurable via the operating system sysctl interface. This should allow the aggressiveness of the congestion control mechanism to be simply tuned. This section provides an overview of this element of the CDG mechanism, which we explore in Section III.

A. CDG algorithm

Most delay-based CCs infer congestion based on an estimate of the path queuing delay; whether or not it is above or below a preset threshold. This works well when the path propagation delay can be measured accurately, however, preexisting queuing delays often cause an underestimate of the actual path queuing delay, leading to situations such as the so-called latecomer advantage problem [1]. CDG, on the other hand, infers congestion by looking at changes in RTT trends.

Originally CDG was designed with heuristics to help it coexist with loss-based CC, something delay-based congestion control mechanisms struggle to do. However, this functionality is not needed for a LBE mechanism. We turn it off, via the sysctl interface, for our investigation into its suitability as a LBE mechanism with dynamically tuneable aggressiveness.

1) Measuring delay trends: CDG maintains two variables $g_{\min,n}$ and $g_{\max,n}$ which represent the change in minimum and maximum RTT respectively, between RTT rounds n and $n-1$. They are computed every RTT round as shown in (1) and (2):

$$g_{\min,n} = \text{RTT}_{\min,n} - \text{RTT}_{\min,n-1}\,, \tag{1}$$

$$g_{\max,n} = \text{RTT}_{\max,n} - \text{RTT}_{\max,n-1}\,. \tag{2}$$

To reduce noise from outlier measures, the gradients are smoothed using a moving average filter as shown in (3):

$$\bar{g} = \sum_{i=n-a}^{n} \frac{g_i}{a} \tag{3}$$

where a is the length of the filter.

2) Probabilistic backoff: CDG probabilistically backs off based on the value of either of these gradients. This helps CDG operate with the noise of the delay signal as well as helping to desynchronize the response to congestion should multiple CDG transports be operating together. The cwnd backoff probability is calculated according to (4):

$$P[\text{Backoff}] = 1 - e^{-\bar{g}/G}\,, \tag{4}$$

where G is a scaling parameter (set to 3 by default). When CDG operates in congestion avoidance mode, $P[\text{Backoff}]$ is calculated as described above every RTT round. If a randomly generated number $X \in [0,1]$ is larger than $P[\text{Backoff}]$, CDG will backoff by doing a multiplicative reduction of cwnd by a factor β (0.7 by default) as described in (5):

$$\text{cwnd} = \begin{cases} \beta \cdot \text{cwnd} & \bar{g} > 0 \wedge X > P[\text{Backoff}]\,, \\ \text{cwnd} + 1 & \text{otherwise.} \end{cases} \tag{5}$$

[1]Centre for Advanced Internet Architectures, Swinburne University of Technology, Australia.

Note that, since it takes one RTT to measure the effect of backing-off, CDG only backs-off at most every second RTT.

B. CDG as a dynamically adjustable LBE congestion control

CDG is already supported in both Linux and FreeBSD as a loadable CC module. Turning off the co-existence heuristics allows CDG to operate in "LBE-mode". Variable delay characteristics (such as base delay and maximum queuing delay) of a path do not cause issues with CDG's LBE characteristics as they do with LEDBAT, because its use of trends in delay rather than inferred queuing delay with thresholds.

A study of CDG as an LBE transport in a home WLAN setting [9] showed that CDG would add no more than 20–40 ms to the 90th percentile RTT, whereas NewReno and CUBIC would add between 110–560 ms to the 90th percentile RTT. Further, it was shown that the queuing delay induced by CDG was constant with variable buffer sizes, while NewReno and CUBIC would grow linearly with increasing buffer sizes.

Although CDG flows with different RTT will grow cwnd at different rates, the exponential nature of Eq. (4) ensures that differences in RTT do not change the TCP fairness between flows. Most importantly, the tunability of the probability scaling parameter G enables CDG to exhibit variable aggressiveness, and thus variable LBE-ness, towards other flows.

Tuning the G parameter of CDG should result in different levels of aggressiveness towards other flows, because it scales the probability $P[\text{Backoff}]$ that a delay gradient signal will make CDG back off. In the limit:

$$\lim_{G\to\infty} P[\text{Backoff}] = \lim_{G\to\infty} 1 - e^{-\bar{g}/G} = 0, \tag{6}$$

and conversely

$$\lim_{G\to 0} P[\text{Backoff}] = \lim_{G\to 0} 1 - e^{-\bar{g}/G} = 1. \tag{7}$$

As G increases CDG the probability of backing off due to delay-gradient congestion indications decreases, in the limit becoming 0 and making CDG as aggressive as its underlying Reno congestion control mechanism reacting to packet loss. As G decreases CDG becomes less and less aggressive.

III. Experimental evaluation

In this section we evaluate the efficacy of using the G parameter to tune the aggressiveness of CDG. We look at the share of capacity two CDG flows receive with respect to two competing CUBIC flows.

A. Experimental setup

Our test-bed setup, depicted in Fig. 1, consists of two sender and two receiver machines connected in a dumbbell topology via two bridge machines, with a router machine in between. All machines run Debian distributions with Linux kernel version 4.2. To emulate typical delays in wide area networks, the two bridge machines artificially delay packets using `netem`. The *router* acts as a bottleneck, shaping incoming traffic to 10 Mbps with the `tc` utility[2]. The tail-drop router buffer

[2]Both `tc` and `netem` are part of the Linux `iproute2` utility package.

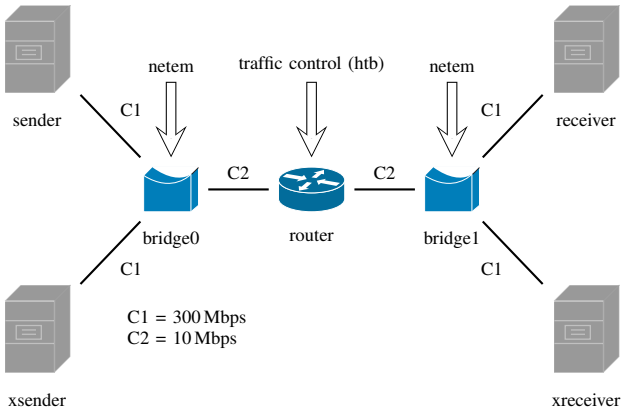


Figure 1: Testbed topology.

is sized, depending on emulated delay, to one bandwidth-delay product (BDP) worth of packets[3]. All machines are interconnected via Ethernet cables and switches running at 300 Mbps. The *sender* machine is configured to use CDG for CC to send variable LBE traffic towards the *receiver* machine, while the *xsender* machine uses the Linux default CUBIC CC to send BE traffic towards the *xreceiver* machine. We use the `iperf` tool[4] to generate greedy traffic for both the LBE and BE flows.

Random UDP background traffic is generated between each sender–receiver pair to prevent emulation artifacts, such as synchronization, corrupting the results. Background traffic is sent using the D-ITG traffic generator [10]. 1450-byte packets are sent with an exponentially distributed inter-departure time of 23.2 ms from each sender, producing on average about 1 Mbps at the bottleneck.

Data was collected at the interfaces of the *sender* and *xsender* machines using `tcpdump`[5]. All experiments have been repeated 10 times.

B. CDG vs. CUBIC throughput share

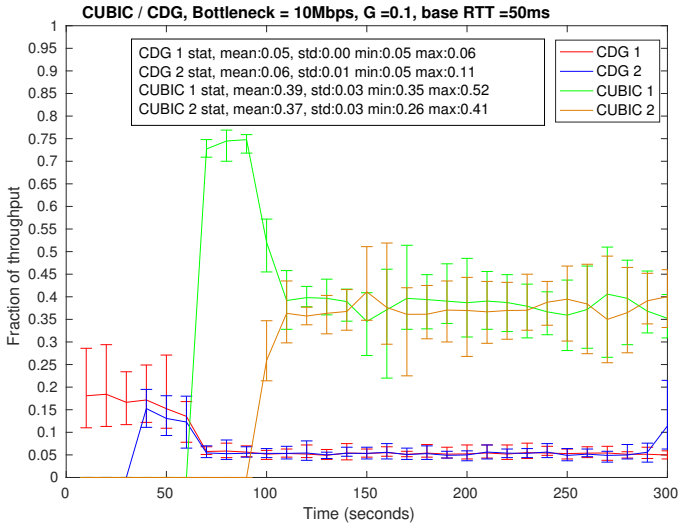
In this set of experiments flows start up with a $30 + X$ s gap between them, where X is a uniformly distributed random number in the range $[0, 5]$ s. The CDG flows start first, followed by the CUBIC flows. Each run lasts 5 minutes, with 10 runs for each combination of G and RTT. The average throughput is calculated from the last 2.5 minutes of the experiment. Error bars span the minimum and maximum observations. Note that CDG's co-existence heuristics are disabled in these experiments. All CDG parameters other than G are set to their Linux defaults. 20 values of G are chosen in the range $[0.1, 120]$[6]. 7 RTT values are used in the range $[20, 300]$ ms.

[3]The MTU is set to 1500 bytes.

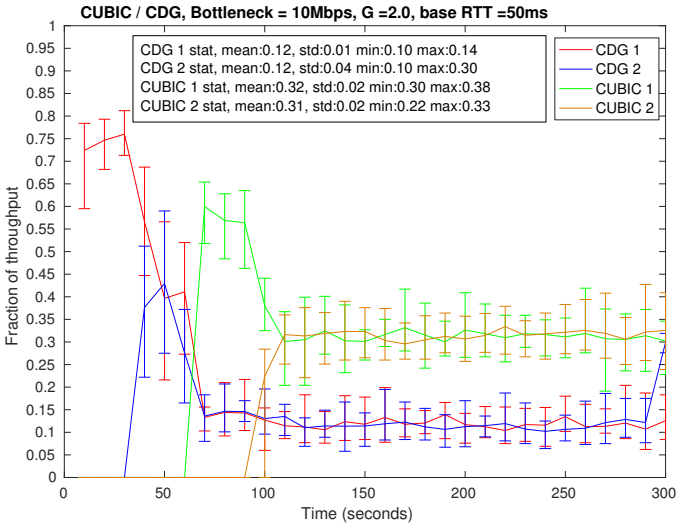
[4]We used version 2.0.5, available from: https://iperf.fr/download/ubuntu/iperf_2.0.5+dfsg1-2_amd64.deb.

[5]http://www.tcpdump.org/release/tcpdump-4.6.2.tar.gz

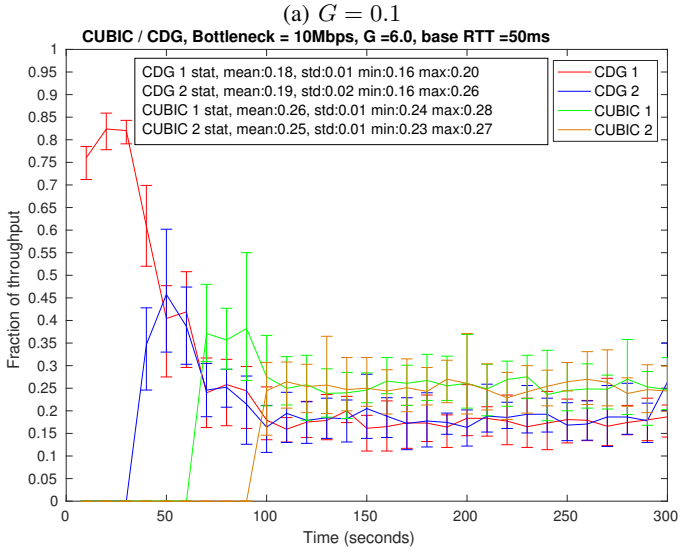
[6]0.002 and 120 are the lowest and highest permitted values of G in the Linux CDG implementation respectively. However, experiments indicate that the difference between G=0.1 and G=0.002 is negligible in most cases.



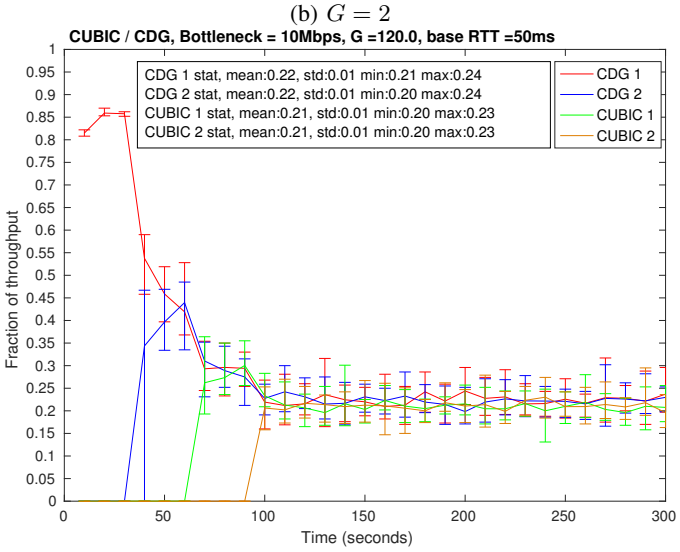
(a) $G = 0.1$



(b) $G = 2$



(c) $G = 6$



(d) $G = 120$

Figure 2: CDG vs. CUBIC share of throughput

Figure 2 shows sample time series plots, with summary statistics, of throughput share between CDG and CUBIC flows, when base RTT (RTT_{base}) is set to 50 ms, for G values of 0.1, 2, 6 and 120. When G is small (see Fig. 2a), the aggregate throughput of the CDG flows stabilize at quite a small fraction of the available capacity before the CUBIC flows start. This is due to the high probability of backing off due to delay-gradient based congestion indications, even when queuing delay is relatively small. When the CUBIC flows start, CDG yields to the CUBIC flows, and maintains a throughput share of ≈5% (see box to the left of the legend with statistics) capacity per flow on average when $G = 0.1$. From $G = 2$ (see Figs. 2b to 2d) CDG is able to fully utilize the available capacity before the CUBIC flows start. The amount of capacity CDG yields to the CUBIC flows decreases as G increases. This culminates in an almost even relative share between CDG and CUBIC flows when $G = 120$ for $\text{RTT}_{\text{base}} = 50$ ms (see Fig. 2d).

We now look at how RTT_{base} influences the relative share between CDG and CUBIC flows for different values of G. Figure 3 summarizes the experiments, showing the relationship between CDG's throughput share against CUBIC (Note that the x-axis in Fig. 3 is neither linear nor logarithmic). We see the total fraction of throughput achieved by CDG flows when the G parameter is varied for different values of RTT_{base}.

Looking at the 20 ms curve in Fig. 3, we see that when RTT_{base} is low, CDG's throughput can be tuned to take between 10% and 50% of capacity per flow. CDG reacts faster to changes in delay, but also increments the cwnd more frequently, which means smaller adjustments to G have larger impact compared to when RTT_{base} is high. As RTT_{base} increases, so does the lower bound on CDG's share, while the upper bound remains limited to Reno. Also, the shape of the curves goes from being concave, where adjustments to G have highest impact for smaller values of G, to convex, where

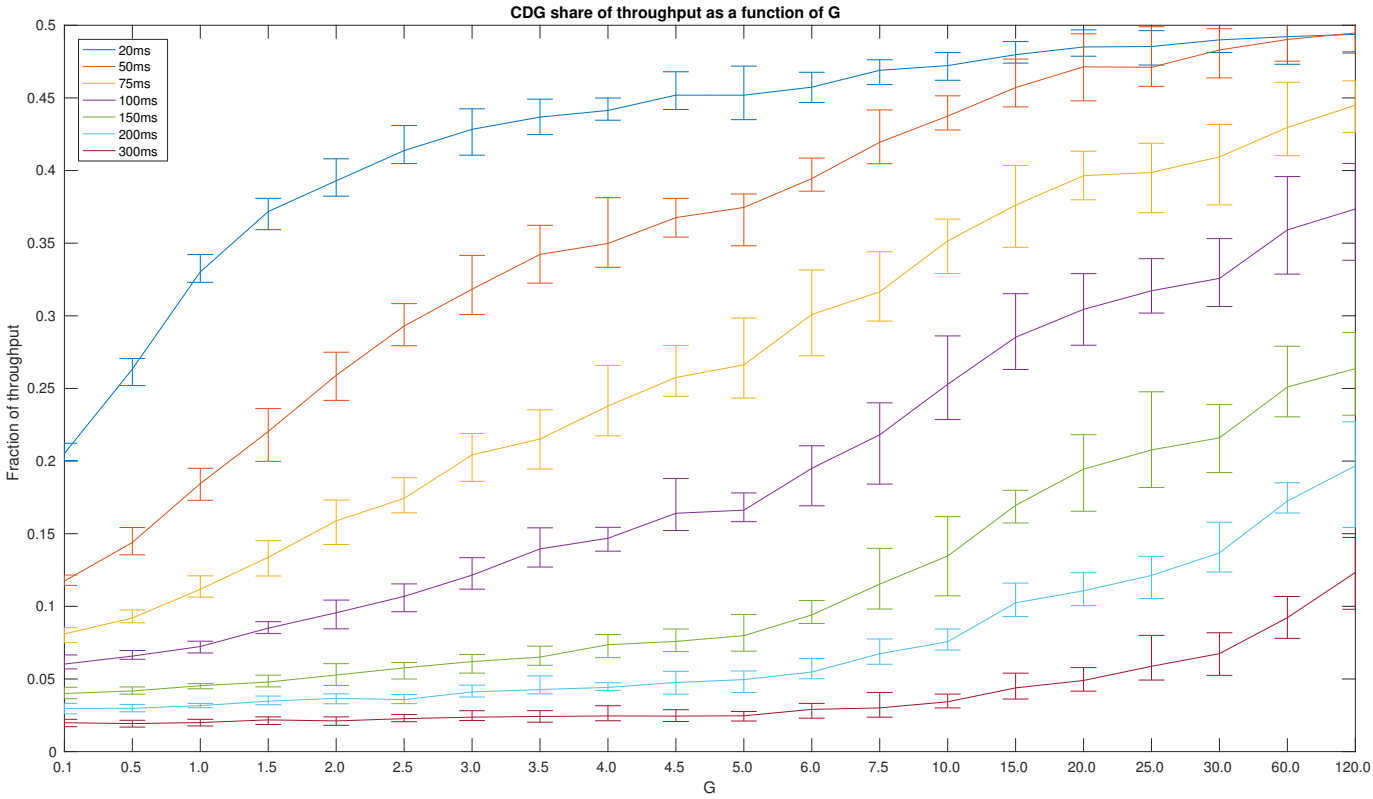


Figure 3: Aggregate throughput of two CDG (LBE) flows for varying values of G, when running against two CUBIC flows. Note that the G scale is not linear, nor is it logarithmic. Error bars span minimum and maximum observations.

G must be set to higher values to have an effect on CDG aggressiveness. This is due to the fact that CDG is competing with a high-speed TCP variant (i.e., CUBIC), CDG's window increase rate is inversely proportional to RTT_{base}, incrementing cwnd maximally once per RTT in congestion avoidance mode as shown in (5). On the other hand, CUBIC's cwnd growth function relies on time between packet loss events and is thus more or less independent of RTT_{base}. Consequently, CDG's tuning range—i.e., the difference between the lowest and highest tunable share of throughput—will be limited by RTT_{base}.

Note however that this does not affect the LBE property of CDG. Because CDG is limited to be no more aggressive than Reno, a CDG LBE flow should not utilize more than (roughly) half of the capacity even if G is set to the maximum value.

Given that CUBIC is roughly insensitive to the RTT, in cases where RTT is low Reno may grow its cwnd faster than CUBIC normally would. To keep up with RTT-dependent CCs CUBIC keeps another window emulating Reno window growth for cases where delay is low. If the emulated window is larger than the CUBIC window, CUBIC's cwnd is increased by the difference between the emulated Reno window and its own cwnd.

Table I summarizes the tuning range of CDG for different values of RTT_{base}. The lower bound assumes only one CDG flow illustrated in Fig. 3. The upper bound assumes the number of CUBIC flows are less than or equal to the number of CDG flows. We see from the table that the tuning range of CDG is dependent on RTT_{base}. When $\text{RTT}_{\text{base}} \leq 75$ ms the range is wide. As RTT increases beyond 75 ms the range shrinks, because the lower bound converges against its limit while the upper bound decreases faster. The range is thinnest at $\text{RTT}_{\text{base}} = 300$ ms. A deadline-aware LBE service based on CDG will benefit from taking its RTT_{base} into consideration when calculating an appropriate value of G.

Table I: Minimum and maximum percent share of capacity achieved by CDG flows, when competing with equivalent CUBIC flows for different values of RTT_{base} (see Fig. 2).

Base RTT (ms)	Tuning range, as min-max % of share
20	10.3–49 %
50	6.7–49 %
75	4.05–44.5 %
100	3.0–37.3 %
150	2.0–26.3 %
200	1.5–19.6 %
300	1.0–12.3 %

IV. Conclusion and Future Work

A transport service that allows its aggressiveness to be tuned, provides a base for implementing a deadline-aware LBE service. Key to providing an LBE service is reacting quickly to changes in network conditions. Generally path delay more quickly indicates congestion build up than packet loss or

standard ECN, making delay-based CC a good choice for an LBE mechanism. We identify CDG, a delay-gradient CC, as a candidate base for deadline-aware LBE for this reason, along with the ease to tune its aggressiveness with the G scaling parameter, and because it does not rely on knowing path delay characteristics such as RTT_{base} or the maximum queuing delay to maintain LBE behavior. Tuning the aggressiveness with respect to an approaching deadline requires some sort of control mechanism. Hayes *et al.* [11] propose a framework for adding LBE behavior with deadline awareness to *any* congestion control mechanism. This framework can be easily applied to CDG and is a future work direction.

We have evaluated its suitability through a set of emulated experiments comparing the Linux versions of CDG and CUBIC over a variable RTT path. Our results indicate that the Linux implementation of CDG can be tuned via its G parameter to achieve a range of between 20–100% of what a competing CUBIC flow would achieve under the same conditions with an RTT of 20 ms. The aggressiveness tuning range of CDG with respect to CUBIC is dependent on RTT. This dependence stems from CDG's cwnd increase policy, which by default limits cwnd growth to 1 MSS (maximum segment size) per RTT. In cases where end-to-end latency is high, this may impose limits on the guarantee that a task will be completed within a specified deadline. One possible solution may be to use CDG's experimental *alpha_inc* parameter to allow cwnd to increase at a rate higher than 1 MSS per RTT in these circumstances. Another way of overcoming this limitation may be to combine a high-speed TCP window growth mechanism, such as the one used by CUBIC, with the delay-gradient based congestion indications of CDG. Study of these possibilities is left for future work.

We note that this study is restricted to comparisons with CUBIC and that the future Internet may use new and quite different congestion controls. The effect this will have on tuning the G parameter is an item for further study, though we note that the framework proposed in [11] gives some direction as to how the operation of a DA-LBE machanism may be adapted in such an environment. In our evaluation we have disabled CDG's loss-tolerance heuristic. Future work could investigate whether CDG's loss-tolerance heuristic is useful for an LBE transport over lossy links. Also, in our experiments we assume a FIFO tail drop queuing discipline at the bottleneck queue. Active Queue Management methods like CoDel [12] and PIE [13], which try to limit maximum queuing delay, could affect a delay-based CC's estimates of congestion and consequently limit its tunable aggressiveness. Future work will therefore evaluate CDG's LBE range against delay-limiting AQM schemes.

Acknowledgment

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644334 (NEAT). The views expressed are solely those of the author(s).

We also thank the anonymous reviewers for their helpful suggestions to improve the paper.

References

[1] D. Ros and M. Welzl, "Less-than-best-effort service: A survey of end-to-end approaches", *IEEE Communications Surveys and Tutorials*, vol. 15, no. 2, pp. 898–908, May 2013.

[2] S. Shalunov, G. Hazel, J. Iyengar and M. Kuehlewind, *Low Extra Delay Background Transport (LEDBAT)*, RFC 6817 (Experimental), Internet Engineering Task Force, Dec. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6817.txt.

[3] D. Rossi, C. Testa, S. Valenti and L. Muscariello, "LEDBAT: The New BitTorrent Congestion Control Protocol", in *2010 Proceedings of 19th International Conference on Computer Communications and Networks*, Aug. 2010, pp. 1–6.

[4] G. Carofiglio, L. Muscariello, D. Rossi and S. Valenti, "The quest for LEDBAT fairness", in *Proc. IEEE GLOBECOM*, Miami, Dec. 2010.

[5] D. Ros and M. Welzl, "Assessing LEDBAT's Delay Impact", *IEEE Communications Letters*, May 2013.

[6] S. Trang, E. Lochin, C. Baudoin, E. Dubois and P. Gelard, "FLOWER – Fuzzy Lower-than-Best-Effort Transport Protocol", in *IEEE Conference on Local Computer Networks*, Oct. 2015.

[7] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to end congestion avoidance on a global Internet", *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.

[8] D. Hayes and G. Armitage, "Revisiting TCP Congestion Control using Delay Gradients", in *IFIP Networking Conference*, May 2011.

[9] G. Armitage and N. Khademi, "Using Delay-Gradient TCP for Multimedia-Friendly 'Background' Transport in Home Networks", in *IEEE Conference on Local Computer Network*, Oct. 2013.

[10] A. Botta, A. Dainotti and A. Pescapè, "A tool for the generation of realistic network workload for emerging networking scenarios", *Computer Networks*, vol. 56, no. 15, pp. 3531–3547, 2012.

[11] D. A. Hayes, D. Ros, A. Petlund and I. Ahmed, "A framework for less than best effort congestion control with soft deadlines", in *IFIP Networking Conference (IFIP Networking)*, to be published, Jun. 2017.

[12] K. Nichols and V. Jacobson, "Controlling Queue Delay", *ACM Queue*, vol. 10, no. 5, 20:20–20:34, May 2012.

[13] R. Pan, P. Natarajan, F. Baker and G. White, *Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem*, RFC 8033 (Experimental), Internet Engineering Task Force, Feb. 2017. [Online]. Available: http://www.ietf.org/rfc/rfc8033.txt.