Revisiting TCP Congestion Control using Delay Gradients

David A. Hayes and Grenville Armitage
{dahayes,garmitage}@swin.edu.au

Centre for Advanced Internet Architectures
Swinburne University of Technology, Melbourne, Australia

Abstract. Traditional loss-based TCP congestion control (CC) tends to induce high queuing delays and perform badly across paths containing links that exhibit packet losses unrelated to congestion. Delay-based TCP CC algorithms infer congestion from delay measurements and tend to keep queue lengths low. To date most delay-based CC algorithms do not coexist well with loss-based TCP, and require knowledge of a network path's RTT characteristics to establish delay thresholds indicative of congestion. We propose and implement a delay-gradient CC algorithm (CDG) that no longer requires knowledge of path-specific minimum RTT or delay thresholds. Our FreeBSD implementation is shown to coexist reasonably with loss-based TCP (NewReno) in lightly multiplexed environments, share capacity fairly between instances of itself and NewReno, and exhibits improved tolerance of non-congestion related losses (86 % better goodput than NewReno in the presence of 1 % packet losses).

1 Introduction

A key goal of TCP (transmission control protocol) [23] is to expedite the reliable transfer of byte-streams across the IP layer's packet-based service while minimizing congestion inside both end hosts and the underlying IP network(s) [2]. Congestion control (CC) for TCP is a challenging, yet practical, research topic attracting interest from both academia and industry [9].

Traditional *loss-based* TCP CC considers IP packet loss to indicate network or end-host congestion. The source retransmits the lost packet *and* briefly slows its transmission rate. Yet TCP's own probing for a path's maximum capacity will also induces packet losses when a path's capacity is reached.

This is broadly reasonable where internet traffic flows over link layers with low intrinsic bit error rates (such as wires or optical fibers) and the traffic is largely loss-tolerant. However, loss-based CC is an increasingly *un*reasonable solution. Today's internet encompasses a mix of TCP-based loss-tolerant and UDP-based loss-sensitive traffic (such interactive online games or Voice over IP) flowing over a mixture of fixed and wireless link layer technologies (such as 802.11-based wireless LANs, 802.16 WiMAX last-mile services, IEEE 802.15.4 ZigBee wireless links to smart energy meters, and so on). Wireless link layers tend to exhibit packet losses that are unrelated to congestion.

In 1989 Jain suggested an alternative CC approach where congestion along an end to end path is inferred from measurements of round trip time (RTT, or *delay*) [15]. CC based on delay measurements can optimise transmission rates without inducing packet losses, and offers the potential to be insensitive to packet losses that are not being caused by congestion.

Many variations have emerged since [15] (as noted in Section 2). However, most of these are what we refer to as *delay-threshold* algorithms – they infer congestion when path delays hit or exceed certain thresholds. Unfortunately, meaningful thresholds are hard to set if little is known about the network path that packets will take. In addition, competing delay-threshold flows can suffer relative unfairness if their inability to accurately estimate a path's base RTT (the smallest possible RTT for the path) leads to thresholds established on the basis of different estimates of base RTT [19].

We propose a novel *delay-gradient* CC technique, and implement it in FreeBSD 9.0. Our approach exhibits improved tolerance to non-congestion related packet losses, and improved sharing of capacity with traditional NewReno TCP [10] in *lightly multiplexed* environments (such as home Internet scenarios). Unlike typical delay-threshold CC techniques, we do not rely on a priori knowledge of a path's minimum or typical RTT levels to infer the onset of congestion.

The rest of our paper is structured as follows. Section 2 summarises key delay-based CC algorithms and their issues. Section 3 describes our delay-gradient CC algorithm. Section 4 covers our implementation, experimental analysis and results, while future work and conclusions are covered in sections 5 and 6.

2 Background

In this section we summarise past efforts at measuring and interpreting network layer delay for TCP CC, and differentiating between congestion- and non-congestion related packet loss.

2.1 Using network delay for TCP congestion control

Proposals to use network delay for TCP congestion control rely on there being a correlation between delay and congestion. Although some studies have shown a low correlation between loss events and increases in RTT [20], this is not an obstacle to CC since it is the aggregate behavior of flows which is important [22].

Proposals differ in the way they measure delay (RTT, one way delay, per packet measurements, etc), how they infer congestion (set thresholds, etc), and how they adjust the sender's congestion window (`cwnd`) in response to congestion. In the following descriptions, β is the multiplicative decrease factor, θ represents a delay threshold, i^{th}RTT measurement $= \tau_i$, smallest RTT $= \tau_{\min}$, largest RTT $= \tau_{\max}$, and the i^{th} one way delay $= d_i$.

Jain's 1989 CARD (Congestion Avoidance using Round-trip Delay) algorithm [15] utilises the normalized delay gradient of RTT, $\left(\frac{\tau_i - \tau_{i-1}}{\tau_i + \tau_{i-1}}\right) > 0$, to infer

congestion, and Additive Increase Multiplicative Decrease (AIMD, $\beta = \frac{7}{8}$) to adjust `cwnd`. DUAL [27] uses $\tau_i > \frac{(\tau_{\min}+\tau_{\max})}{2}$ with delay measured every 2^{nd} RTT, and AIMD ($\beta = \frac{7}{8}$) to adjust `cwnd`.

Vegas [6] uses $\tau_i > \tau_{\min} + \theta$ (normalized by the data sent), delay measured every RTT and Additive Increase Additive Decrease (AIAD) to adjust `cwnd`. Fast TCP [28], TCP-Africa [16] and Compound TCP (CTCP) [26] all infer congestion similarly to Vegas, but use smoothed RTT measurements and differ in how they adjust `cwnd` upon detecting congestion.

TCP-LP [17] uses $\overline{d}_i > d_{\min} + \delta(d_{\max} - d_{\min})$, based on smoothed one way delay (d) using TCP time stamps, and AIMD with a minimum time between successive window decreases to adjust `cwnd`. Probabilistic Early Response TCP (PERT) [3] uses dynamic thresholds based on inferred queuing delay ($q_j = \tau_j - \tau_{\min}$) using smoothed RTT measurements and probabilistic reaction to queuing delay inspired by Random Early Discard (RED), with loss probability matching when $q_j \geq 0.5 q_{\max}$.

Hamilton Delay [7, 18] ("HD", a CC algorithm from the Hamilton Institute), and our own variant of HD [14], implement probabilistic adjustment of `cwnd` based on queuing delay, RTT thresholds and a backoff function.

2.2 Differentiating congestion and non-congestion related loss

Using delay to infer congestion opens the possibility to differentiate between congestion and non-congestion related packet losses. For example, Biaz and Vaidya [5] investigated techniques based on TCP Vegas rate, normalised throughput gradients and normalised delay gradients, but found them to be inadequate. Cen et al. [8] investigated a number of proposals including inter-arrival time [4], a threshold based RTT scheme (Spike), a statistical RTT based scheme (ZigZag), and hybrids of these. Recently Zhao et al. [29] proposed WMTA that uses comparative loss rates of small and large packets to infer whether a wireless link is suffering wireless or congestion related losses. All of these techniques provide insight into the problem, but fall short of a robust and accurate solution.

3 Delay-gradient TCP congestion control

In this section we describe how CDG ("CAIA Delay-Gradient") modifies the TCP sender in order to: (a) use the delay gradient as a congestion indicator, (b) have an average probability of back off that is independent of the RTT, (c) work with loss-based congestion control flows (such as NewReno), and (d) tolerate non-congestion packet loss, but backoff for congestion related packet loss. First we answer the question of why it is important to revisit CC based on delay gradients, and then we describe our proposed delay-gradient algorithm.

3.1 Why use delay gradient?

Inspired by Jain's CARD [15], we have two reasons for revisiting the use of delay gradient for delay-based congestion control.

First, delay-threshold CC algorithms typically require an accurate estimate of a path's base (smallest possible) RTT in order to properly share capacity among themselves, and ensure network queuing delay stays low [19]. Second, choosing thresholds is difficult. The right compromise between queuing delay and network utilisation requires knowing each flow's path – a challenge if flows traversing differing numbers of hops are to compete fairly for available capacity.

These limitations make delay-threshold algorithms, on their own, problematic for Internet-wide deployment. In contrast, delay-gradient CC relies on *relative* movement of RTT, and adapts to particular conditions along the paths each flow takes.

3.2 Delay gradient signal

RTT is a noisy signal – a cleaner signal is required for inferring congestion from the gradient of RTT over time. CDG uses the maximum RTT ($\tau_{\max}$) seen in a measured RTT interval[1], along with the minimum RTT ($\tau_{\min}$) seen within a measured RTT interval. Based on these, two measures of gradient (change in RTT measurement per RTT interval) are kept, where n is the nth RTT interval:

$$g_{\min,n} = \tau_{\min,n} - \tau_{\min,n-1} \quad (1) \qquad g_{\max,n} = \tau_{\max,n} - \tau_{\max,n-1} \quad (2)$$

The maximum and minimum measurements are less noisy than per packet RTT measurements. Nevertheless we apply the moving average smoothing of equation 3, which may be calculated iteratively using equation 4 (where a is the number of samples in the moving average window[2]).

$$\bar{g}_n = \sum_{i=n-a}^{n} \frac{g_i}{a} \quad (3) \qquad \bar{g}_n = \bar{g}_{n-1} + \frac{g_n - g_{n-a}}{a} \quad (4)$$

where $g_i = g_{\min,i}$ for calculating $\bar{g}_{\min,n}$ or $g_i = g_{\max,i}$ when calculating $\bar{g}_{\max,n}$.

We implemented an enhanced RTT measuring module [13] to obtain live measurements without the noise caused by duplicate acknowledgments, segmentation offload and sack (see [22] for more on RTT sampling).

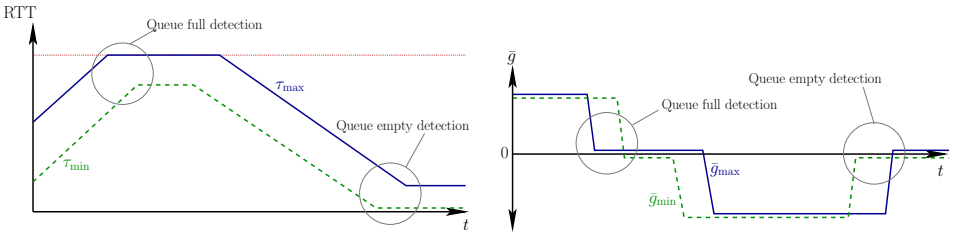
An alternative to the moving average would be exponential smoothing. However, if the measured $g_{\max,n}$ ceased to grow because a queue along the path was full, an exponential average would only approach $\bar{g}_{\max,n} = 0$ in the limit. A moving average would achieve $\bar{g}_{\max,n} = 0$ in a samples.

3.3 Differentiating congestion and non-congestion related loss

In order to tolerate the kinds of packet loss common in, say, wireless environments, we must infer whether or not packet loss is related to congestion. For simple drop tail queues, congestion related loss is due to overflow of a queue along the packet's path. To infer such events CDG uses both $\bar{g}_{\min}$ and $\bar{g}_{\max}$.

[1] The same time interval that is used in Vegas [6].

[2] Although the probabilistic backoff described in 3.4 can provide sufficient smoothing, moving average smoothing helps with the loss tolerance and coexistence heuristics. When operating in slow start mode $g_{\min,n}$ and $g_{\max,n}$ are used without smoothing for a more timely response to the rapid w increases.



(a) Idealised RTT dynamics for queue full and empty events

(b) Idealised gradient dynamics for queue full and empty events. ($\bar{g}_{\min}$ and $\bar{g}_{\max}$ are vertically offset for clarity.)

Fig. 1: Queue full and queue empty scenarios, highlighting the detection areas

Figure 1a illustrates our assumption that when a queue fills to capacity, $\tau_{\max}$ stops increasing before $\tau_{\min}$ stops increasing, and that the reverse is true for a queue moving from full to empty. Figure 1b shows the idealised gradients for these two conditions (with the lines for $\bar{g}_{\min}$ and $\bar{g}_{\max}$ offset slightly for clarity). Based on this CDG estimates the state of the path queue to be $Q \in \{\text{full, empty, rising, falling, unknown}\}$. Only when $Q = \text{full}$ are packet losses treated as congestion signals.

3.4 RTT independent backoff

We use Equation 5 as a probabilistic backoff mechanism to achieve fairness between flows having different base RTT.

$$P[\text{backoff}] = 1 - e^{(\bar{g}_n / G)} \tag{5}$$

were $G > 0$ is a scaling parameter and $\bar{g}_n$ will either be $\bar{g}_{\min,n}$ or $\bar{g}_{\max,n}$. Our implementation uses a lookup table for e^x and a configurable FreeBSD kernel variable for G.

The exponential nature of $P[\text{backoff}]$ means that on average a source with a small RTT which sees smaller differences in the RTT measurements will have the same average $P[\text{backoff}]$ of a source with a longer RTT which will see larger differences in the RTT measurements.

3.5 Congestion window progression

In congestion avoidance mode, CDG updates the congestion window (w) once every RTT according to Equation 6

$$w_{n+1} = \begin{cases} w_n \beta & X < P[\text{backoff}] \wedge \bar{g}_n > 0 \\ w_n + 1 & \text{otherwise} \end{cases} \tag{6}$$

where w is the size of the TCP congestion window in packets[3], n is the nth RTT, $X = [0, 1]$ is a uniformly distributed random number, and β is the multiplicative

[3] CDG increments a byte-based w by the maximum segment size every RTT.

decrease factor ($\beta = 0.7$ in our testbed experiments). Since the effect of this update will not be measured in the next RTT interval, the next calculation of $g_{\min}$ or $g_{\max}$ is ignored. Thus a delay-gradient congestion indication will cause CDG to back off at most once every two RTT intervals[4].

In slow start mode w increases identically to NewReno. The decision to reduce w and enter congestion avoidance mode is made per RTT as per Equation 6, or on packet loss as in NewReno, whichever occurs first.

3.6 Competing with loss-based CC

CDG uses two mechanisms to mitigate the loss of fair share of available capacity when sharing bottleneck queues with loss-based TCP flows: *ineffectual backoff detection*, and a loss-based *shadow window* from [14].

Ineffectual backoff detection If CDG backs off multiple times, b, due to delay-gradient congestion indications, but $\bar{g}_{\min}$ or $\bar{g}_{\max}$ are still not negative, then CDG presumes that its backoffs have been ineffectual because it is competing with a loss based flow. Consequently, CDG does not back off due to delay-gradient congestion indications for b' further delay gradient congestion indications unless either $\bar{g}_{\min}$ or $\bar{g}_{\max}$ become negative in the process. (In our CDG implementation both b and b' are configurable FreeBSD kernel variables.)

Shadow window CDG recovers some of its lost sending capability by utilising the shadow window idea from [14] to mimic the loss based backoffs of TCP NewReno. The shadow window (s) is initialised as follows:

$$s_{i+1} = \begin{cases} \max(w_i, s_i) & \text{delay based backoff} \\ 0 & Q = \text{empty} \\ s_i & \text{otherwise} \end{cases} \tag{7}$$

If delay-gradient triggers a backoff then $s_{i+1} = \max(w_i, s_i)$, but if CDG guesses a bottleneck queue has emptied then $s_{i+1} = 0$, otherwise s is unchanged.

3.7 Window update on packet loss

If a packet loss occurs, the congestion window (w) is updated as follows:

$$w_{i+1} = \begin{cases} \frac{\max(s_i, w_i)}{2} & Q = \text{full} \wedge \text{packet loss} \\ w_i & \text{otherwise} \end{cases} \tag{8}$$

In the case of packet losses, the multiplicative decrease factor is 0.5 (as in NewReno), and w is set to half the bigger of s (the shadow window) and w.

Using the shadow window concept from [14] improves CDG's coexistence with loss based flows. We do not reclaim the lost transmission opportunities, but this approach does lessen the impact of the extra delay-gradient based backoffs. Figure 2 gives and example illustrating how this works.

[4] TCP Vegas [6] uses a similar idea during slow start

Referring to the regions indicated by circled numbers:

1. w grows as normal for TCP congestion avoidance (one packet per RTT)
2. Delay-gradient congestion indication meeting Equation 6's criteria, s is initialised to w, then $w = \beta w$
3. w continues to react to delay-gradient congestion indications
4. s shadows NewReno-like growth
5. A packet loss occurs (Q = full), so w is set to $s/2$ rather than $w/2$ (per Equation 8)

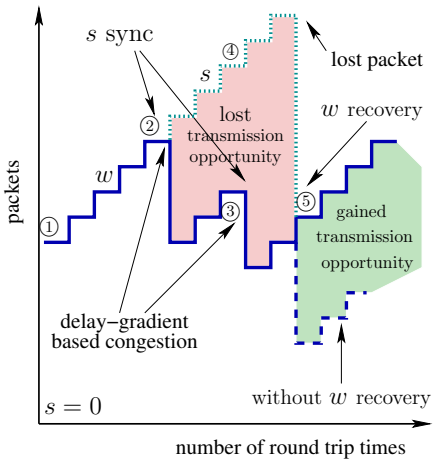


Fig. 2: Behaviour of shadow window (s) and congestion window (w) when competing with loss-based TCP flows.

4 Experimental Analysis

Here we present our experimental evaluation of CDG, primarily focusing on:

- Tolerance of NewReno and CDG to non-congestion related losses
- Sharing dynamics between three homogeneous flows (CDG or NewReno)
- Competition of up to two NewReno flows and up to two CDG flows
- Sharing dynamics between two homogeneous flows of different RTTs

The "source" hosts of Figure 3 implement CDG as a FreeBSD 9.0 kernel module [1], whilst the "sinks" are unmodified FreeBSD hosts. We used Giga-

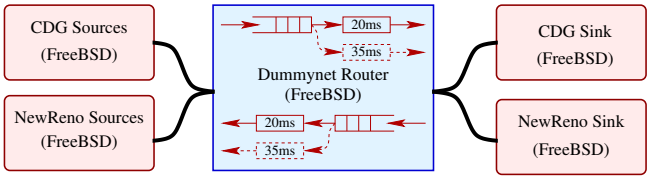


Fig. 3: Experimental Testbed

bit Ethernet links to the dummynet-based [24] router. This router provides a 10 Mbps bottleneck link and base RTTs of 40 ms (20 ms each way) and 70 ms (35 ms each way) as needed. The bottleneck queue is 84 packets long, corresponding to a maximum queuing delay of about 100 ms with 1500 byte packets.

TCP traffic is generated using Netperf (`http://www.netperf.org/`). NewReno flows use the default parameters. CDG operates with moving average sample window $a = 8$, exponential scaling parameter $G = 3$, ineffectual backoff trigger $b = 5$ and ineffectual backoff ignore count $b' = 5$. The non-congestion related loss is implemented as random packet loss added by dummynet in the the forward (data) path only. Each experiment is repeated 10 times. Where appropriate, graphs show the 20^{th}, 50^{th}, and 80^{th} percentiles (marker at the median, and error bars spanning the 20^{th} to 80^{th} percentiles).

4.1 Tolerance to non-congestion related losses

First we look at the impact of non-congestion related losses on TCP *goodput*[5]. Figure 4 shows the average goodput achieved over 60 s versus the probability of non-congestion related packet loss for New Reno, TCP Vegas [6], and CDG. We also show the theoretical maximum throughput under loss conditions given by the $B = \frac{\text{pkt\_size}}{\text{rtt}} \frac{C}{\sqrt{p}}$ model proposed by Mathis et al. [21] (where B is the expected throughput, rtt = 40 ms, p is the probability of packet loss, and $C = \sqrt{\frac{3}{2}}$).

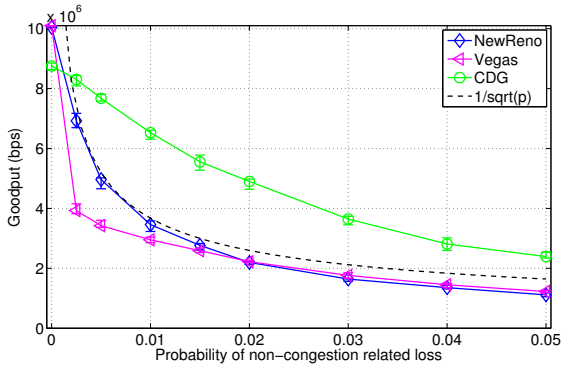


Fig. 4: Goodput of NewReno, Vegas and CDG with non-congestion losses

NewReno and Vegas goodput decreases markedly with non-congestion related losses, tracking [21]'s 1/sqrt(p) curve. Vegas reacts to both loss and delay as congestion signals. High speed TCP variants, such as CUBIC[12], which reduce w by less on packet loss and increase w much more quickly during congestion avoidance than NewReno can recover from packet loss more quickly than NewReno. This capability in CDG is not tested here for fair comparison with NewReno. Note that Compound TCP [26] performs worse than NewReno at these levels of non-congestion packet loss (Compound TCP begins performing slightly worse than NewReno when losses exceed about 0.5 %).

CDG is noticeably better at tolerating non-congestion losses. Although still reacting to loss, CDG's use of delay-gradient information improves its ability to infer whether any given loss event is due to congestion along the path. CDG is conservative, preferring to have a false positive than a false negative. Nevertheless, CDG's goodput still drops as loss rates increase. It spends proportionally more time retransmitting lost packets and less time growing `cwnd` (Equation 6's w) as CDG does not increment `cwnd` during the recovery process.

4.2 Homogeneous Capacity Sharing

Here we contrast the way NewReno and CDG share capacity with instances of their own 'type'. Each experiment uses three 60 s NewReno or CDG flows sharing the bottleneck link, with the first, second and third flows starting at 0 s, 20 s and

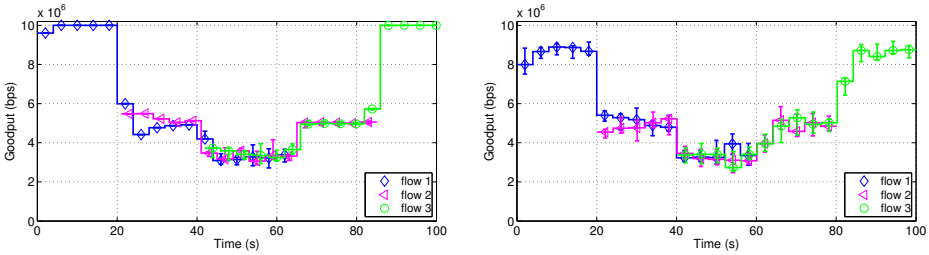
[5] Usable data transferred per unit time, excluding retransmitted payloads

40 s respectively. We examine the case where all losses are due to congestion, and then artificially add an extra 1% packet loss rate unrelated to congestion.

Figure 5 (goodput over time) shows that both NewReno and CDG share quite fairly among themselves when there is no non-congestion related packet loss. Figure 6 (the path RTT over time) reveals that NewReno induces far higher, and more oscillatory, queuing delays than CDG.

Each NewReno flow (Figure 5a) takes longer to converge on their fair share of link rate than the equivalent CDG flow (Figure 5b). This is because NewReno consistently pushes RTT up to 140 ms (40 ms base RTT plus 100 ms queuing delay when queue is full), so its feedback loop cannot react to the onset of self-induced congestion during slow start as quickly as CDG's feedback loop.

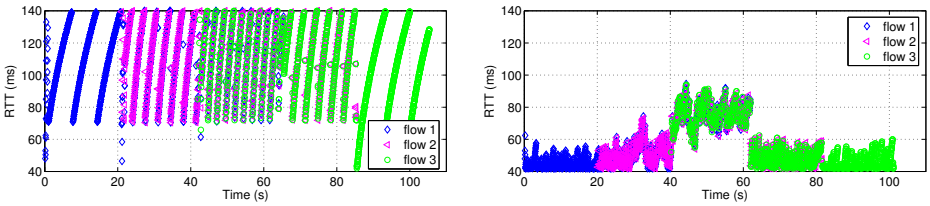
CDG can allow the link to become idle for brief periods of time, so a single CDG flow will never quite match the best goodput of a NewReno flow in the absence of non-congestion related losses (Figure 5 between $t = 10$ s and $t = 20$ s).



(a) Three NewReno flows sharing link

(b) Three CDG flows sharing link

Fig. 5: Homogeneous capacity sharing between three flows on a link having no non-congestion related losses. Flows begin at 20 s intervals and transmit for 60 s. Goodput averaged every 4 s with the point at the middle of the 4 s interval.

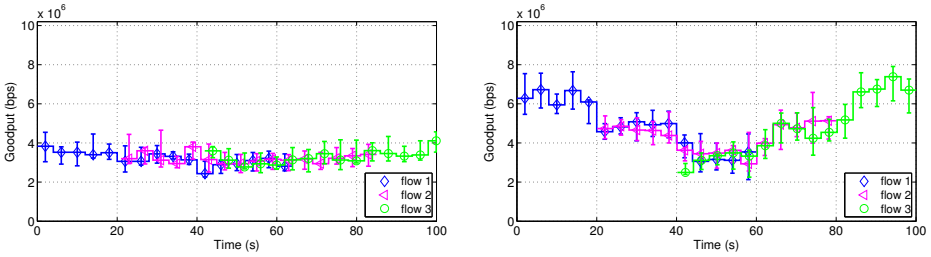


(a) NewReno RTT dynamics due to induced queuing delays

(b) CDG RTT dynamics due to induced queuing delays

Fig. 6: Path RTT versus time during the trials in Figure 5. (The number of points has been reduced by a factor of 20 for clarity.)

CDG's relatively benign impact on RTT is likely to be attractive to other applications sharing a congestion point. In contrast, NewReno cyclically fills the queue until packet loss occurs (Figure 6a). (In similar scenarios CUBIC induces similar or higher average delays than NewReno [25]. Compound TCP's congestion window is always `cwnd`+`dwnd`, where `dwnd` uses a Vegas-style delay calculations, so it will also not induce lower queuing delays than NewReno.)



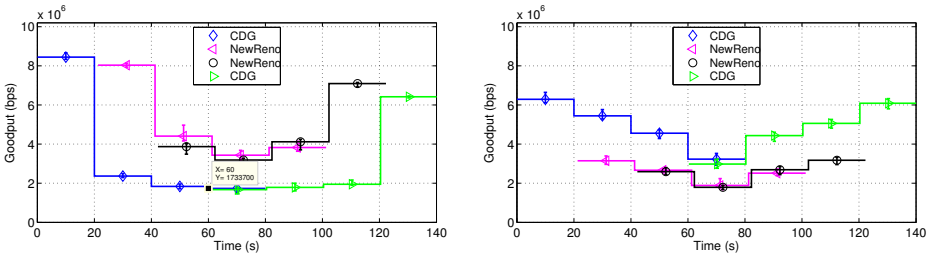
(a) Three NewReno flows sharing link (b) Three CDG flows sharing link

Fig. 7: Homogeneous capacity sharing between three flows on a link with a 1 % random probability of non-congestion related losses. Flows begin at 20 s intervals and transmit for 60 s. 4 s averages, with the point in the middle of the 4 s interval.

Figure 7 repeats the experiment with 1 % probability of additional packet loss (unrelated to congestion). The additional losses dominate and cripple NewReno flows, but CDG (cf. Figure 4) continues to utilise and share the available capacity.

4.3 Competing with NewReno

Practical deployment of delay-based CC algorithms is made difficult by the need to coexist with loss-based flows that tend to cyclically overfill queues. We start four 80 s flows (CDG, NewReno, NewReno, and CDG) at 20 s intervals to demonstrate how CDG's use of the shadow window (Sections 3.6 and 3.7 and Figure 2) helps it compete with NewReno for available capacity.



(a) No non-congestion losses. (b) 1 % random non-congestion loss.

Fig. 8: Coexistence between NewReno and CDG – Goodput averaged every 20 s (plotted point is in the middle of the averaging period).

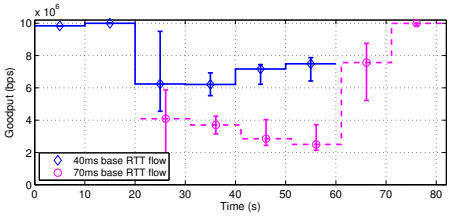
Figure 8a shows coexistence where congestion is the only source of packet loss. The first CDG flow retains about 24 % of the available capacity once the first NewReno flow starts up and takes the rest. This is mainly due to CDG's conservative coexistence heuristic which at first does reduce `cwnd` before the shadow window adjustment is made. CDG does slightly better as the next NewReno and CDG flows join, but never quite claims its fair share.

Figure 8b shows how the dynamics change when there is a 1 % probability of non-congestion packet loss. NewReno's sensitivity to packet loss prevents it from fully utilising the available capacity. In contrast, CDG's intrinsic tolerance

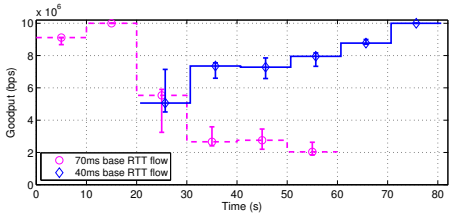
to non-congestion losses allows it to utilise more of the capacity that NewReno is unable to use. (CDG does not capture capacity at the expense of NewReno.)

4.4 Competition between flows having different base RTTs

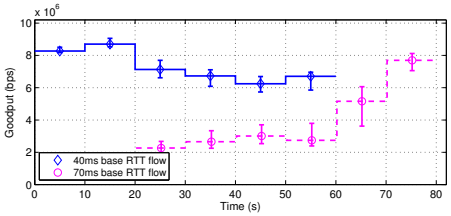
Finally we briefly explore capacity sharing between flows using the same CC algorithm but having different base RTTs (40 ms and 70 ms). Figure 9 shows the goodput results (each point showing goodput averaged over 10 s) for NewReno and CDG in two scenarios: (a) the source with 40 ms base RTT starts first (S-L), and (b) the source with 70 ms base RTT starts first (L-S).



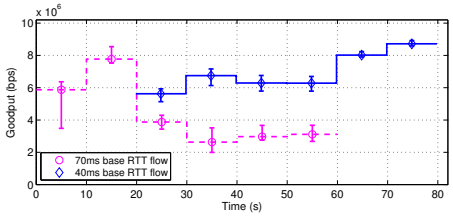
(a) Two NewReno flows sharing a link – shorter RTT flow starting first (S-L)



(b) Two NewReno flows sharing a link – longer RTT flow starting first (L-S)



(c) Two CDG flows sharing a link – shorter RTT flow starting first (S-L)



(d) Two CDG flows sharing a link – longer RTT flow starting first (L-S)

Fig. 9: Homogeneous capacity sharing between two flows with different base RTTs, with no non-congestion related losses. Each flow transmits for 60 s, with the second flow starting 20 s later. 10 s averages, with the point in the middle of the 10 s interval.

In both S-L and L-S cases the NewReno and CDG flows with higher (70 ms) base RTT end up with a smaller (roughly 30 %) share of available capacity. The most notable difference between NewReno and CDG is the speed with which capacity sharing stabilises in each case. As noted in Section 4.2, NewReno induces much higher overall RTT and thus its feedback loop reacts more slowly (compared to CDG) to the addition or removal of a competing flow.

In such lightly multiplexed environments with little noise, phase effects can dramatically alter the function of traditional TCP and the resulting throughput [11]. This can lead to larger error bars (particularly evident in Figures 9a,b). CDG does not suffer as much from these effects due to its probabilistic backoff.

5 Further work

We have considered CDG in lightly multiplexed environments where congestion is dominated by a single router (such as might exist with home Internet connections). More work is required to characterise the utility of CDG in highly multiplexed environments, and multi-hop multi-path environments.

CDG's `cwnd` increase mechanism could adopt a more aggressive approach: increasing during the loss recovery mechanism when the loss is deemed to not be due to congestion to better cope with higher loss rates, and increasing `cwnd` more quickly during congestion avoidance than traditionally allowed (relying on CDG's ability to infer congestion before packets are lost).

We explore Delay-gradient CC because it *does not* require accurate knowledge of a path's base RTT. Future work might explore whether combining delay-gradient and absolute queuing delay congestion signals could create a more robust CC algorithm.

6 Conclusion

We have proposed, implemented (under FreeBSD 9.0) and demonstrated CDG – a novel sender-side delay-gradient TCP congestion control algorithm that requires no changes to existing TCP receivers. CDG avoids a key limitation of delay-threshold CC algorithms – their need to establish an accurate measure of a path's base RTT and set thresholds based on actual network path delay characteristics. CDG's improved tolerance to non-congestion related packet losses makes it attractive over paths containing links with non-negligible intrinsic packet loss rates (such as wireless links). CDG can coexist with loss-based TCPs such as NewReno, though it achieves less than its fair share of the capacity in such cases.

CDG uses the maximum RTT and minimum RTT gradient envelope to estimate whether loss is congestion or non-congestion related. CDG subsequently exhibited improved tolerance to non-congestion related losses, with a single CDG flow achieving 65 % of the available capacity at 1 % packet loss, compared to 35 % for NewReno (a 86 % improvement over NewReno). CDG's utilises [14]'s NewReno-like shadow window to help it compete with loss-based TCP CC algorithms.

7 Acknowledgments

This work was made possible in part by a grant from the Cisco University Research Program Fund at Community Foundation Silicon Valley.

References

[1] NewTCP project tools (Aug 2010). `http://caia.swin.edu.au/urp/newtcp/tools.html`, [Accessed 2 December 2010]

[2] Allman, M., Paxson, V., Stevens, W.: TCP Congestion Control . RFC 2581 (Proposed Standard) (Apr 1999). http://www.ietf.org/rfc/rfc2581.txt, updated by RFC 3390

[3] Bhandarkar, S., Reddy, A. L. N., Zhang, Y., Loguinov, D.: Emulating AQM from end hosts. In: SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 349–360. ACM, New York, NY, USA (2007)

[4] Biaz, S., Vaidya, N.: Discriminating congestion losses from wireless losses using inter-arrival times at the receiver. In: Application-Specific Systems and Software Engineering and Technology, 1999. ASSET '99. Proceedings. 1999 IEEE Symposium on, pp. 10–17 (1999)

[5] Biaz, S., Vaidya, N. H.: Distinguishing congestion losses from wireless transmission losses,. In: Seventh International Conference on Computer Communications and Networks (IC3N). New Orleans (Oct 1998)

[6] Brakmo, L. S., Peterson, L. L.: TCP Vegas: end to end congestion avoidance on a global internet. IEEE J. Sel. Areas Commun. 13(8), pp. 1465–1480 (Oct 1995)

[7] Budzisz, L., Stanojevic, R., Shorten, R., Baker, F.: A strategy for fair coexistence of loss and delay-based congestion control algorithms. IEEE Commun. Lett. 13(7), pp. 555–557 (Jul 2009)

[8] Cen, S., Cosman, P. C., Voelker, G. M.: End-to-end differentiation of congestion and wireless losses. IEEE/ACM Trans. Netw. 11(5), pp. 703–717 (2003)

[9] Floyd, S.: Congestion Control Principles. RFC 2914 (Best Current Practice) (Sep 2000). http://www.ietf.org/rfc/rfc2914.txt

[10] Floyd, S., Henderson, T., Gurtov, A.: The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 3782 (Proposed Standard) (Apr 2004). http://www.ietf.org/rfc/rfc3782.txt

[11] Floyd, S., Jacobson, V.: On traffic phase effects in packet-switched gateways. Internetworking: Research and Experience 3(3), pp. 115–156 (Sep 1992)

[12] Ha, S., Rhee, I., Xu, L.: CUBIC: A new tcp-friendly high-speed tcp variant. ACM SIGOPS Operating System Review 42(5), pp. 64–74 (2008)

[13] Hayes, D.: Timing enhancements to the FreeBSD kernel to support delay and rate based TCP mechanisms. Tech. Rep. 100219A, Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia (19 February 2010). http://caia.swin.edu.au/reports/100219A/CAIA-TR-100219A.pdf

[14] Hayes, D. A., Armitage, G.: Improved coexistence and loss tolerance for delay based TCP congestion control. In: 35th Annual IEEE Conference on Local Computer Networks (LCN 2010). Denver, Colorado, USA (Oct 2010)

[15] Jain, R.: A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks. SIGCOMM Comput. Commun. Rev. 19(5), pp. 56–71 (1989)

[16] King, R., Baraniuk, R., Riedi, R.: TCP-Africa: An adaptive and fair rapid increase rule for scalable TCP. In: IEEE INFOCOM 2005, pp. 1838–1848 (2005)

[17] Kuzmanovic, A., Knightly, E.: TCP-LP: low-priority service via end-point congestion control. IEEE/ACM Trans. Netw. 14(4), pp. 739–752 (Aug 2006)

[18] Leith, D., R.Shorten, G.McCullagh, J.Heffner, L.Dunn, F.Baker: Delay-based AIMD congestion control. In: Proc. Protocols for Fast Long Distance Networks. California (2007)

[19] Leith, D., Shorten, R., McCullagh, G., Dunn, L., Baker, F.: Making available base-rtt for use in congestion control applications. Communications Letters, IEEE 12(6), pp. 429 –431 (Jun 2008)

[20] Martin, J., Nilsson, A., Rhee, I.: Delay-based congestion avoidance for tcp. IEEE/ACM Trans. Netw. 11(3), pp. 356–369 (Jun 2003)

[21] Mathis, M., Semke, J., Mahdavi, J., Ott, T.: The macroscopic behavior of the tcp congestion avoidance algorithm. SIGCOMM Comput. Commun. Rev. 27(3), pp. 67–82 (1997)

[22] McCullagh, G., Leith, D. J.: Delay-based congestion control: Sampling and correlation issues revisited. Tech. rep., Hamilton Institute – - National University of Ireland, Maynooth (2008)

[23] Postel, J.: Transmission Control Protocol. RFC 793 (Standard) (Sep 1981). `http://www.ietf.org/rfc/rfc793.txt`, updated by RFC 3168

[24] Rizzo, L.: Dummynet: a simple approach to the evaluation of network protocols. ACM SIGCOMM Computer Communication Review 27(1), pp. 31–41 (1997)

[25] Stewart, L., Armitage, G., Huebner, A.: Collateral damage: The impact of optimised TCP variants on real-time traffic latency in consumer broadband environments. In: Proceedings of IFIP/TC6 NETWORKING 2009. Aachen, Germany (May 2009)

[26] Tan, K., Song, J., Zhang, Q., Sridharan, M.: A compound TCP approach for high-speed and long distance networks. In: INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings, pp. 1–12 (Apr 2006)

[27] Wang, Z., Crowcroft, J.: Eliminating periodic packet losses in the 4.3-Tahoe BSD TCP congestion control algorithm. SIGCOMM Comput. Commun. Rev. 22(2), pp. 9–16 (Apr 1992)

[28] Wei, D. X., Jin, C., Low, S. H., Hegde, S.: FAST TCP: Motivation, architecture, algorithms, performance. IEEE/ACM Trans. Netw. 14(6), pp. 1246–1259 (Dec 2006)

[29] Zhao, H., ning Dong, Y., Li, Y.: A packet loss discrimination algorithm in wireless ip networks. In: Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference on, pp. 1–4. Beijing (Sep 2009)