Using Delay-Gradient TCP for Multimedia-Friendly 'Background' Transport in Home Networks

Grenville Armitage
Centre for Advanced Internet Architectures
Swinburne University of Technology
Melbourne, Australia
garmitage@swin.edu.au

Naeem Khademi
Department of Informatics
University of Oslo
Norway
naeemk@ifi.uio.no

***Abstract*—Home networks are seeing increased deployment of Wireless LAN (WiFi) links between conventional, gigabit/second wired Ethernet segments. This means an increasing number of internal bottlenecks, even as home networks are also expected to support latency-sensitive applications, regular TCP flows and an emerging class of low-priority, time-insensitive 'background' TCP flows. This paper explores the novel use of CDG v0.1 (a delay-gradient TCP) for such background TCP connections in home networks. We show a CDG flow induces latencies of only tens of milliseconds regardless of the bottleneck's internal buffer size (useful when coexisting with latency-sensitive traffic) while achieving a significant fraction of spare link capacity. We also show CDG does not gratuitously steal capacity from commonly deployed "foreground" TCPs such as CUBIC and NewReno.**

I. Introduction

Home networks are increasingly a mix of conventional 100Mbps and 1Gbps wired Ethernet segments and WiFi wireless LAN (WLAN) links. PCs, home file servers, entertainment consoles and handheld devices are just as likely to be communicating amongst themselves over the home's WLAN as they are communicating to and from "the Internet". And despite the increasing diversity of internal bottlenecks, home networks must support both latency-sensitive applications (such as UDP-based VoIP and online games) and bulk data transfer applications (such as TCP-based email, web surfing, peer to peer file transfers, streaming video, and so forth).

A key challenge is that common *loss-based* TCP congestion control (CC) algorithms (such as NewReno [1] and CUBIC [2]) tend to cause cyclical filling and draining of a network bottleneck's buffers. A side-effect of such CC behaviour is that all traffic sharing the bottleneck will experience additional end to end latency due to increased *queueing delays* [3]. Such delays are problematic for concurrent latency-sensitive applications, particularly given the past decade's growth of available buffer space (*bufferbloat*) almost everywhere a queue might form (routers, switches, device drivers, and so forth) [4]. Loss-based TCPs also increase delays over WLAN links by being insensitive to contention for transmission slots.

There is also an emerging class of applications whose data transfers can be categorised as low-priority, time-insensitive *background* TCP flows (such as automated in-house backup systems, peer to peer file transfers, cache pre-fetching, off-peak software updates, and so forth). Such applications require TCP-style reliable data transfer but tolerate deferring to other traffic. This has prompted development of "background", "scavenger" or "less than best effort" styles of TCP service [5].

This current paper explores the novel use of CDG v0.1 [6] (a delay-gradient TCP) for background TCP connections in home networks. We show that in lightly-multiplexed scenarios CDG adds latencies down in the tens of milliseconds regardless of the bottleneck's internal buffer size while achieving a significant fraction of available link capacity. We further show CDG does not gratuitously steal capacity from commonly deployed "foreground" TCPs such as CUBIC and NewReno in WLAN environments.

The rest of this paper is structured as follows. Section II provides a brief review of loss-based and delay-based TCPs, the challenges of WLAN links, and the limitations of existing approaches to background TCP. CDG itself is summarised in Section III. Section IV illustrates CDG coexisting with loss-based TCPs in a WLAN environment. We conclude with Section V.

II. Background

Here we briefly review how loss-based TCP interacts with latency-sensitive traffic, the challenges of WLAN links, the ability of delay-based TCPs to minimise queuing delays and the limitations of existing approaches to background TCP.

A. How regular TCP interacts with latency-sensitive traffic

The capacity probing and congestion control behaviour of loss-based TCP is problematic when sharing a bottleneck with latency-sensitive traffic flows [3], [7]). The issue is how loss-based TCPs fill and drain bottleneck queues.

The number of unacknowledged packets a TCP sender can have in flight is limited by the smaller of the TCP receiver's advertised receive window (`rwnd`, indicating how much data the receiver can currently buffer) and the TCP sender's congestion window (`cwnd` or w, an estimate of how much data the network path can absorb) [8]. Loss-based TCPs grow w when packets are getting through, eventually exceeding the level required to fully utilise the path. Further w growth simply fills the bottleneck's queue, eventually leading to lost packets. The TCP flow's w then shrinks, the queue drains (although not back to empty) and TCP's probing begins again.

The increasing availability of buffer space in network devices and end-hosts means TCP receivers are able to advertise increasingly large `rwnd`, and TCP senders grow w to fill the bottleneck's bigger buffers. The resulting queuing delays impact on all traffic traversing a bottleneck queue – these have been observed as high as multiple seconds over consumer paths [9], [10].

B. 802.11 Wireless LANs (WLANs)

WLAN-connected laptops, smartphones, tablets, games consoles, network extenders and so forth are increasingly common in the modern home, generating a mix of TCP-based and latency-sensitive traffic flows. Delays over WLAN links originate both from buffering (within stations) and contention (between stations competing for radio transmission slots). Contention levels go up with the number of concurrent hosts, increasing the latencies experienced by all parties.

When TCP traffic is directed through an access point (AP) towards WLAN clients, the AP can coordinate downstream transmission of all TCP Data packets. The upstream (reverse) direction will be corresponding ACK packets. With n stations and $n < 100$ we usually see ≤ 3 stations causing upstream contention by sending ACKs at the same instant [11], [12].

Concurrent TCP flows in the upstream direction are more troublesome [10], [13], [14]. Congestion manifests itself as collisions (contention for upstream transmission slots) as each station's TCP grows w independently. This in turn causes localised queue buildup in each station's upstream interface. A full queue at one station provides no feedback to other stations, who keep contending for transmission slots, leading to higher latencies than in the downstream case. CC algorithms that grow w more aggressively than NewReno (such as CUBIC, which is the default in Linux) compound the problem.

WLANs also continuously adapt their modulation and coding schemes in response to wireless noise, interference and contention levels. Examples include schemes such as SampleRate [15] and Minstrel [16] for rate adaption (RA) [13]. Transient drops in bit-rate also translate to increased transmission delays.

C. Delay-based TCP and reduced queuing delays

Proposals for *delay-based* congestion indications have been around since Raj Jain's CARD (Congestion Avoidance using Round-trip Delay) in 1989 [17]. By observing trends in RTT, delay-based TCPs infer the onset of congestion when a bottleneck queue is *beginning* to fill rather than after it fills and loses packets. Since queues need not fill, queuing delays are kept low regardless of each bottleneck's available buffering.

Subsequent variations (such as DUAL [18], Vegas [19], Fast TCP [20], TCP-Africa [21], Compound TCP (CTCP) [22] and 'CAIA HD' [23]) have emerged since [17], differing in their delay measurements (RTT, one-way delay, per-packet measurements, etc), how they set thresholds to infer congestion, and how they adjust w in response to inferred congestion (proportional, probabilistic, etc).

Unfortunately, *delay-threshold* variants (CC algorithms that infer congestion when absolute delay reaches certain thresholds) find meaningful thresholds hard to set if little is known about a flow's network path characteristics over time. In addition, competing delay-threshold flows can suffer relative unfairness if their inability to accurately estimate a path's base (smallest possible) RTT leads to thresholds established on the basis of different estimates of base RTT [24].

D. Background / Low-priority TCP

Here we briefly look at the techniques and limitations of existing approaches to "background" TCP approachs.

1) Prior work: An early example is "TCP Nice" [25] which uses delay-threshold CC behaviour until RTT (τ) reaches a pre-configured fraction t of the bottleneck's inferred queue size. When a certain number of packets in one RTT are observed to have $\tau > t$, congestion is assumed and w is halved. Otherwise, Vegas-style w control is used. Furthermore, Reno-style w control is used if packet loss is detected.

TCP LP ("Low priority") [26], [27] aims to utilise excess bandwidth yet defer quickly to regular TCP flows. Such flows are detected when TCP LP's smoothed one-way delay (OWD) measurement (d_i) exceeds a pre-configured threshold $\overline{d}_i > d_{\min} + \delta(d_{\max} - d_{\min})$, indicating the onset of bottleneck congestion caused by other traffic.

ImTCP-bg [28] uses both regular estimates of available bandwidth along the path and an enhanced RTT-based mechanism for congestion indications.

LEDBAT ("Low Extra Delay Background Transport" [29]) aims to use available excess bandwidth while limiting the increase in induced path delays and yielding quickly in the presence of competing flows. Its congestion signal relies on changes in d_i measurements up to a specified delay threshold. While applicable to TCP, LEDBAT's algorithm has also been built into proprietary UDP-based peer to peer applications.

2) Limitations: To date background TCPs for home networks have not exhibited low induced delays when competing with latency-sensitive flows.

TCP Nice needs regular $\tau_{\min}$ (smallest RTT) and $\tau_{\max}$ (largest RTT) measurements to set t as a fraction of estimated queue size. Observing meaningful $\tau_{\max}$ requires one or more flows to regularly push the bottleneck queue to its limit (negatively impacting any concurrent latency-sensitive traffic).

TCP LP needs bottleneck queuing delays to exceed a pre-configured threshold somewhere between recently measured $d_{\min}$ and $d_{\max}$ values, suggesting the need for regular queue filling events to establish $d_{\max}$. ImTCP-bg requires pre-configured thresholds to determine how $\tau_{\min}$ and smoothed mean τ influence w reduction. LEDBAT requires implementation changes in both sender and receiver, and by default allows itself to induce up to $d_{min} + 100ms$ (where $d_{\min}$ is LEDBAT's estimate of the unloaded path's OWD). However, recent work shows that LEDBAT can become confused about $d_{\min}$, and push a path's OWD much higher than $100ms$ [30].

III. CAIA Delay Gradient (CDG) v0.1

Our focus is the novel application of the sender-side *CDG* ("CAIA Delay Gradient") v0.1 TCP [6] for time-insensitive, background data transfers. CDG is a sender-side algorithm compatible with all conventional TCP receivers. By using *relative* variations in RTT to infer congestion it eliminates the need to know a path's minimum, maximum or typical RTT levels, and eliminates reliance on pre-configured delay thresholds. Consequently CDG also coexists well with common types of latency-sensitive traffic (such as VoIP or online games) which do not cyclically push bottlenecks into congestion and/or packet loss.

Here we summarise key elements from [6].

A. A delay gradient signal

CDG tracks the envelope ($\tau_{\max}$ and $\tau_{\min}$) of enhanced RTT estimates [31] seen within a measured RTT interval. Taking n as the nth RTT interval, CDG constructs two gradient signals $g_{\max,n}$ and $g_{\min,n}$ from successive $\tau_{\max}$ and $\tau_{\min}$:

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

In slow start mode $g_{\min,n}$ and $g_{\max,n}$ are used as-is for a timely response to rapid w increases. In congestion avoidance mode CDG uses $\bar{g}_{\min,n}$ and $\bar{g}_{\max,n}$ (gradients smoothed using an iterative moving average, see Equation 4 in [6]).

B. Delay-gradient control of w

In congestion avoidance mode, CDG increments w by one MSS once every RTT unless the smoothed RTT gradient is positive, in which case there is a finite probability CDG will back off. These delay-gradient back-offs are expressed succinctly in Equation 2

$$w_{n+1} = \begin{cases} w_n \beta_d & X < P[\text{back-off}] \wedge \bar{g}_n > 0 \\ w_n + 1 & \text{otherwise} \end{cases} \quad (2)$$

with w in packets, n the nth RTT interval, and β_d the delay-gradient back-off factor ($\beta_d = 0.7$ in [6]). $X = [0, 1]$ is a uniformly distributed random number, and the $X < P[\text{back-off}]$ term (see Equation 5 in [6]) provides an *RTT-independent probabilistic back-off* (a source with a small RTT will have the same average $P[\text{back-off}]$ as a source with a longer RTT).

CDG allows no more than one Equation 2-triggered backoff every two RTT intervals (as the impact of a backoff wont be felt in the next RTT interval). If $\bar{g}_{\min}$ or $\bar{g}_{\max}$ are still not negative after backing off $b = 5$ times, CDG presumes it is competing with one or more loss-based flow(s) and suppresses further delay-gradient back-offs until either $b' = 5$ further delay-gradient congestion indications have occurred or one of $\bar{g}_{\min}$ or $\bar{g}_{\max}$ have become negative during this period. (This is termed *ineffectual back-off detection* in [6].)

Growth of w in slow start mode mimics that of NewReno. The decision to reduce w and enter congestion avoidance mode is made per RTT as per Equation 2, or on packet loss as in NewReno, whichever occurs first.

C. Updating w on packet loss

Packet loss triggers two possible outcomes. If the bottleneck queue is presumed full ($\bar{g}_{\max}$ recently dropped to zero before $\bar{g}_{\min}$) the loss indicates congestion. Otherwise the lost packet needs retransmission but w is left unchanged.

When loss is congestion related, CDG adjusts w as follows:

$$w_{i+1} = \begin{cases} \max(s_i, w_i)\beta_l & Q = \text{full} \wedge \text{packet loss} \\ w_i & \text{otherwise} \end{cases} \quad (3)$$

where $\beta_l = 0.5$ (the classic NewReno-like loss-based multiplicative decrease factor), and s is [23]'s concept of a *shadow window* – a NewReno-like congestion window not subject to the delay-gradient back-offs being applied to w. This improves CDG's coexistence with loss-based flows (which are oblivious to CDG's recent delay-gradient back-offs).

IV. CDG in Wireless LAN environments

This section uses CDG v0.1 implemented for FreeBSD [32] to explore scenarios similar to those that may arise in typical home WLAN-based networks. We look at the willingess of CDG to coexist with loss-based TCP flows, and the benefit to loss-sensitive interactive applications of coexisting with CDG rather than NewReno or CUBIC.[1] All CDG parameters are set as per [6] except where noted below.

A. Methodology

Figure 1 depicts our experimental network on the *Emulab* [33] Wireless LAN testbed. We configured two different WLANs on different channels, with access points (APs) connected via two intermediate routers and a 100 Mbps links. Each wireless node was an Intel PIII 600 MHz with 512 MB RAM, an Atheros-based AR5413 802.11 chipset, FreeBSD 9.0 kernel and standard Atheros driver operating in 2.4 GHz 802.11b/g mode for indoor activities.

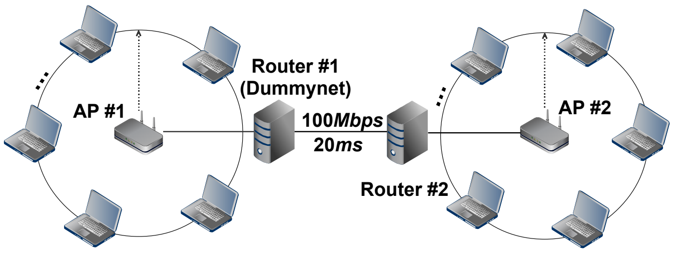


Figure 1: A dumbbell network topology with 802.11 end-nodes

Sending hosts were on one WLAN, receiving hosts on the other. All trial runs were confirmed to have similar channel condition patterns by measuring the RSSI of each frame carrying TCP payload at its respective receiving node.

Dummynet [34] and *ipfw* on Router #1 emulated a wired path between the WLANs having 20 ms one-way delay in each direction (for an $RTT_{base} = 40ms$). With more than 20 ms of

[1]We focus on NewReno and CUBIC as the "background" TCPs from Section II-D are (aside from LEDBAT in a peer to peer client) rarely seen.

buffering at 100 Mbps in Router #1 we ensured the WLANs were the bottlenecks. TCP's send and receive buffers were set large enough to ensure w could grow to fill the bottlenecks. TCP's Maximum Segment Size (MSS) was 1448 bytes. All nodes are synchronized in time using *ntp*.

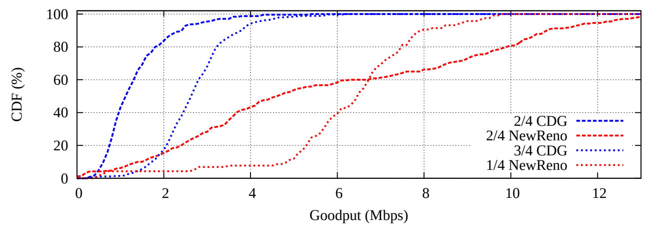
Each experiment is a 60 sec run of TCP traffic from *iperf* repeated 10 times. We captured all traffic using *tcpdump* on the outgoing and incoming interfaces of each station, passively extracted per-packet RTT statistics using the *Synthetic Packet Pairs* (SPP) tool [35], and calculated TCP goodput using *tcptrace* [36]. Coexistence tests with VoIP used *tcpreplay* [37] to inject previously captured VoIP traffic.

B. CDG coexisting with NewReno and CUBIC

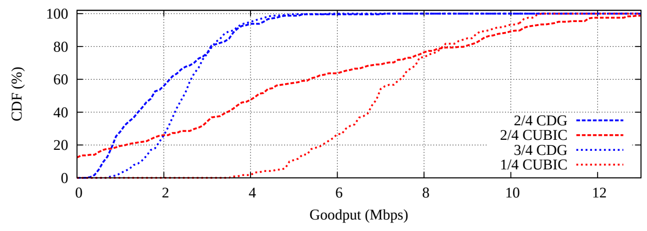
Although CDG v0.1 aims for reasonable performance when CDG flows share bottlenecks with loss-based TCP flows, it will always defer to them in some greater or lesser degree.

Figures 2 and 3 reflect a series of WLAN to WLAN tests with four stations, where either two of four ("2/4") or three of four ("3/4") stations are CDG senders and the rest (one of four, "1/4", or two of four, "2/4") use NewReno or CUBIC.

Figure 2 shows the per-station performance achieved when running CDG with $\beta_d = 0.7$ (per [6]). Figure 2a shows the CDG stations deferring to NewReno stations, with the latter gaining a greater share of the available capacity. Figure 2b similarly shows the CDG stations obtaining a smaller share of capacity than the CUBIC stations.



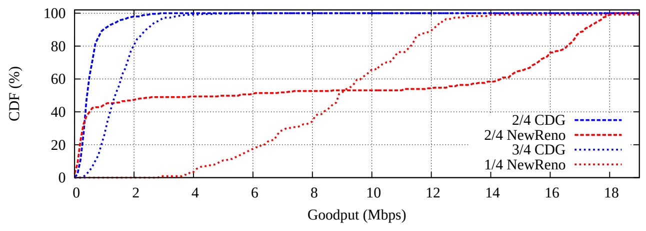
(a) CDG ($\beta_d = 0.7$) coexisting with NewReno



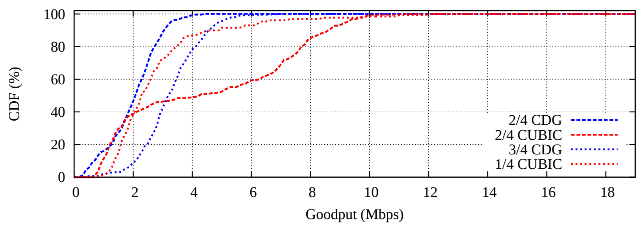
(b) CDG ($\beta_d = 0.7$) coexisting with CUBIC

Figure 2: Per-station goodput, CDG ($\beta_d = 0.7$) coexisting with loss-based TCPs – four WLAN stations either side of an AP

The situation is a little more interesting in Figure 3, when running CDG with a more aggressive delay-gradient backoff of $\beta_d = 0.5$. CDG stations defer to NewReno (Figure 3a) stations even more comprehensively than they did in Figure 2. However, CUBIC (Figure 3b) fails to make good use of its opportunities. This isn't a problem with CDG per se, which (with $\beta_d = 0.5$) provides more transmission opportunities to the CUBIC station(s) than in Figure 2. The issue is that CUBIC's own aggressiveness over the wireless link is detrimental to its own overall performance, particularly when three stations run CDG and one station runs CUBIC.



(a) CDG ($\beta_d = 0.5$) coexisting with NewReno



(b) CDG ($\beta_d = 0.5$) coexisting with CUBIC

Figure 3: Per-station goodput, CDG ($\beta_d = 0.5$) coexisting with loss-based TCPs – four WLAN stations either side of an AP

These distributions of per-station goodputs are a consequence of how CDG responds to contention-based delays induced by NewReno or CUBIC stations. As the loss-based TCP flows grow their congestion windows and push harder, they cause increased contention for wireless transmission slots. The CDG flows perceive the increased RTT and apply delay-gradient back-off, allowing the loss-based flows to consume a larger share of the wireless channel capacity. However, CDG's ineffectual back-off detection ensures the loss-based flows do not completely starve the CDG flows of capacity.

C. Delays induced using CDG, NewReno or CUBIC alone

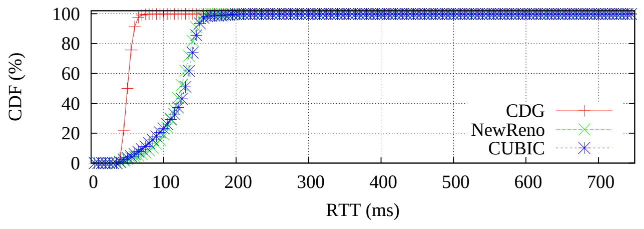
Latency-sensitive applications will always suffer when sharing a congested bottleneck with loss-based TCP flows, whether or not CDG flows are also present. So the next meaningful question is what delays will be imposed on a WLAN network when the *only* other active TCP flows are using CDG with $\beta_d = 0.5$, NewReno or CUBIC.

Our next experiments took eight WLAN stations associated with AP#1 and used iperf to send traffic (upload to) or receive traffic (download from) eight matching *wired* hosts on the other side of Router #1. The WLAN would thus be loaded in the uplink (towards AP#1) or downlink (from AP#1) directions respectively.

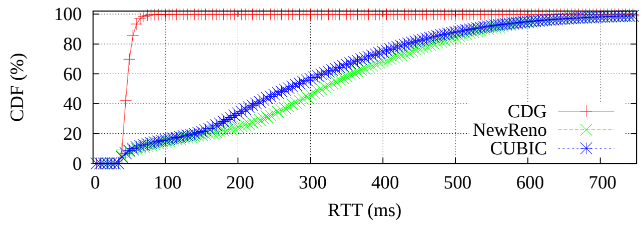
Figure 4a shows the latencies experienced by TCP traffic in the download scenario. CDG induces a 90th-percentile RTT[2] of ~60 ms (only 20 ms over RTT_{base}) whilst NewReno and CUBIC induce 90th-percentile RTTs closer to 150 ms.

The upload scenario in Figure 4b reveals a more dramatic difference. CDG induces a similar 90th-percentile RTT of

[2]We focus on 90th-percentile RTTs because latency-sensitive applications are often more impacted by peak RTTs rather than mean or median RTTs



(a) CDG, NewReno or CUBIC in the download direction



(b) CDG, NewReno or CUBIC in the upload direction

Figure 4: Distributions of induced RTT when running *only* CDG, NewReno or CUBIC (with 40ms base RTT)

~60 ms, but NewReno and CUBIC now induce 90th-percentile RTTs closer to ~600 ms. Loss-based CC algorithms suffer much higher RTTs in the upload scenarios because they increase w without regard for uplink contention delays.

Both figures suggest that latency-sensitive traffic sharing a WLAN with CDG flows would experience RTTs that are both low and consistent (not varying much between minimum and maximum RTT). This is illustrated further in Figure 5 – an extract of induced RTT versus time showing CDG producing consistently low RTT samples while NewReno exhibits the cyclic behaviour common for loss-based CC algorithms.

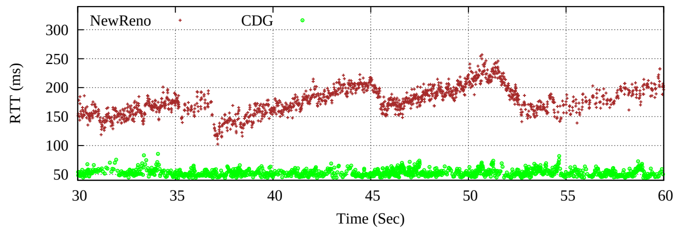


Figure 5: CDG induces lower and more consistent queueing delays over WLAN than NewReno

The distinct difference in RTT distributions is due to CDG being responsive to the additional delay variations introduced at the 802.11 MAC layer (such as increased channel access time and potential collisions). CDG stations adjust their w conservatively, reducing collision probabilities at the MAC layer and leading to lower observed RTTs. In contrast, NewReno and CUBIC increase w without regard for MAC layer contention delays. The detrimental impact on RTT is particularly evident in the upload scenario (Figure 4b).

D. Delay when coexisting with non-reactive VoIP traffic

We next evaluate the impact of CDG, NewReno or CUBIC coexisting with concurrent VoIP-like flows sharing a bottleneck WLAN AP. VoIP traffic tends to be latency-sensitive (due to the nature of the application) yet non-reactive (in that the packet transmission rate usually depends on the voice codec's rate rather than network conditions).

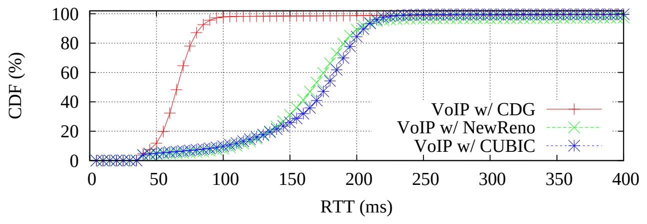
Figure 6 shows the distribution of RTTs experienced by our VoIP-like traffic in two coexistence scenarios:

1) Both senders and receivers are 802.11 nodes (eight stations sending to eight stations through the AP)
2) TCP flows are heading uplink from eight WLAN stations to eight wired destinations

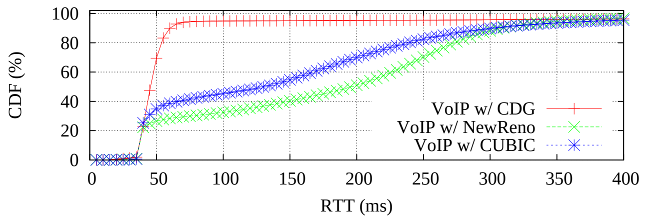
In addition to the TCP flows, each source/destination pair are configured to transfer a 60 sec bi-directional VoIP-like flow consisting of UDP packets in 200 byte Ethernet frames with 20 ms inter-arrival times.

Figure 6a reveals that when both ends are 802.11, our VoIP packets experienced a 90th-percentile RTT of ~80 ms (40 ms over RTT_{base}.) when coexisting with CDG flows. This contrasts with 90th-percentile RTTs of ~200 ms experienced when coexisting with NewReno or CUBIC flows.

Figure 6b reveals that when the traffic runs from WLAN to wired hosts, our VoIP packets experienced a 90th-percentile RTT of ~60 ms when coexisting with CDG flows. This contrasts with 90th-percentile RTTs of ~300 ms our VoIP packets experienced when coexisting with NewReno or CUBIC flows.



(a) 802.11 at both ends of the network path



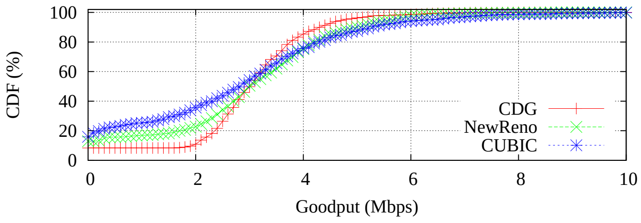
(b) 802.11 stations sending to wired hosts (uplink)

Figure 6: CDF of two-way delay of coexisting VoIP flows

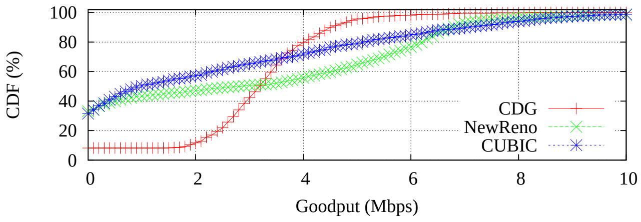
E. Performance when using CDG, NewReno or CUBIC alone

CDG's consistently low RTTs in Section IV-C are only part of the story. Figure 7 shows the CDF of per-station goodput for Section IV-C's download and upload scenarios.

In the download scenario (Figure 7a) applications would achieve similar median goodputs using either NewReno, CUBIC or CDG (3 Mbps, 2.8 Mbps and 3 Mbps respectively). However, the interactions between loss-based CC and upstream wireless contention creates a very different story in the upload scenario (Figure 7b) – applications would achieve median goodputs of 2.9 Mbps, 1 Mbps and 3.2 Mbps per station using NewReno, CUBIC or CDG respectively.



(a) CDG, NewReno or CUBIC in the download direction



(b) CDG, NewReno or CUBIC in the upload direction

Figure 7: Distribution of per-station goodput (per 5 sec intervals) when running *only* CDG, NewReno or CUBIC

CUBIC's poor median performance in Figure 7b is due to its more aggressive w growth behaviour, leading to higher uplink contention and hence a wide range of goodput during each trial. Furthermore, we see that both CUBIC and NewReno flows were often stalling (achieving less than 100Kbps in a 5-second window) – when one flow stumbles from time to time, the loss-based growth of the other seven flows briefly starve the first one of capacity.

In contrast, Figures 7a and 7b suggest background applications using CDG in either direction will experience more consistent goodput compared to using NewReno or CUBIC.

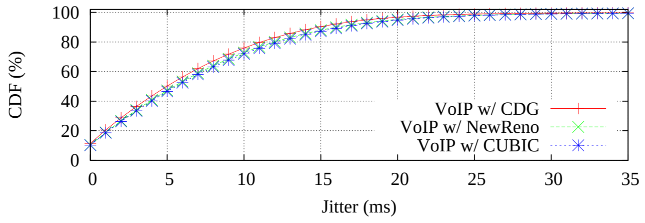
F. Short-term RTT variations are modest

The widely spread RTTs for VoIP coexisting with NewReno or CUBIC in Figure 6 is due to variations over relatively long (multi-second) time frames. Over very short periods (such as from one packet to the next) the RTT tends to change in only small deltas regardless of our choice of CC algorithm.

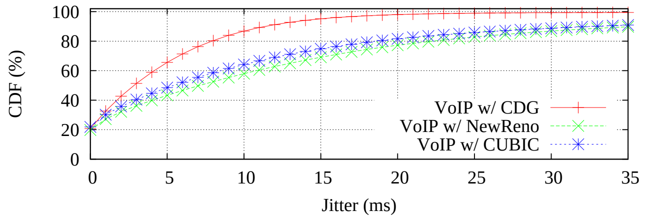
Let us briefly define $\Delta RTT_{perpacket}$ as the difference between the RTTs experienced by two consecutively transmitted VoIP packets. Figure 8a reveals that VoIP will see reasonably low $\Delta RTT_{perpacket}$ regardless of CC algorithm when we have 802.11 at both ends of the path. However, if the 802.11 stations are sending to wired hosts instead, Figure 8b shows that VoIP will experience less $\Delta RTT_{perpacket}$ when coexisting with CDG flows. Latency-sensitive applications may cope acceptably if their play out buffers adapt to long-term changes in RTT while absorbing short-term RTT variations.

G. Independence from bottleneck bufferbloat

We ran a simple experiment using wired hosts either side of a bottleneck node to illustrate CDG keeping buffer utilisation low regardless of a bottleneck's available buffer space. Dummynet provided a 10Mbps bottleneck with 40ms base RTT and buffer sizes of {84,167,250,416,582,831} packets.



(a) 802.11 at both ends of the network path



(b) 802.11 stations sending to wired hosts (uplink)

Figure 8: CDF of $\Delta RTT_{perpacket}$ of coexisting VoIP flows

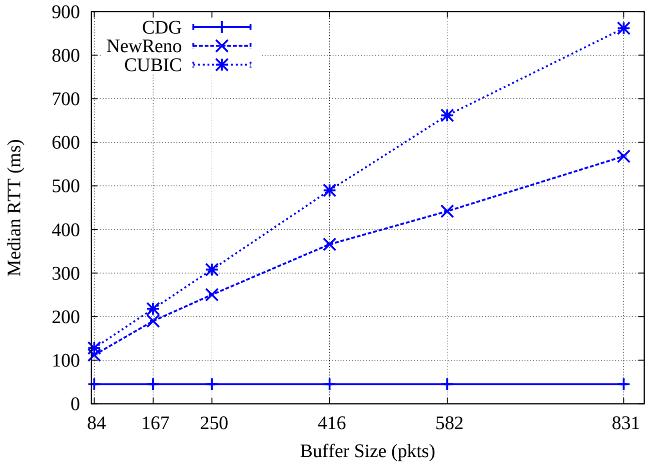


Figure 9: Induced RTT vs bottleneck buffer size

Figure 9 shows the median RTT induced by independent flows of CDG, NewReno and CUBIC versus the bottleneck buffer size. It is clear that latency-sensitive traffic in a mixed wired/wireless home network will not experience large RTT swings if CDG is used for 'background' TCP, regardless of the mix of consumer devices in the network and the varied levels of bufferbloat that may exist.

V. Conclusions and Future Work

Home networks often mix conventional 100Mbps/1Gbps wired Ethernet segments and lower-speed WLAN links. Loss-based TCP algorithms cause cyclical filling and draining of a network bottleneck's buffers, and are insensitive to contention for transmission slots on wireless segments. This can cause significant increase in network delays experienced by all traffic sharing a bottleneck WLAN link within the home.

Using a FreeBSD-based test bed we have demonstrated that CDG v0.1 [6] is a viable choice for applications whose data transfers can be categorised as low-priority, time-insensitive *background* TCP flows. CDG TCP avoids the significant induced queuing delays common to NewReno, CUBIC (and intrinsic to proposed "scavenger" or "less than best effort" styles of TCP such as TCP Nice, TCP LP, or LEDBAT).

In lightly-multiplexed WLAN scenarios we show CDG consistently adding no more than 20 ms to 40 ms to the 90th-percentile RTT under circumstances where NewReno and CUBIC would add between 110 ms and 560 ms, while achieving a significant fraction of spare link capacity. CDG does not gratuitously steal capacity from commonly deployed "foreground" TCPs such as CUBIC and NewReno, although these TCPs may not themselves work well in WLAN environments.

One area of future work is tuning CDG's transitions between delay-gradient and loss-based backoffs to better utilise shared WLAN uplinks, particularly when loss-based TCPs inhabit the same WLAN client.

Acknowledgements

We thank Lawrence Stewart for his help setting up and explaining FreeBSD's pluggable congestion control system and its support for NewReno, CUBIC and CDG. This work was supported in part by a gift from the Cisco University Research Program Fund, a corporate advised fund of Silicon Valley Community Foundation.

References

[1] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm," RFC 6582, Internet Engineering Task Force, April 2012. [Online]. Available: http://tools.ietf.org/html/rfc6582

[2] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, Jul 2008.

[3] L. Stewart, G. Armitage, and A. Huebner, "Collateral damage: The impact of optimised TCP variants on real-time traffic latency in consumer broadband environments," in *Proceedings of IFIP/TC6 NETWORKING 2009*, May 2009.

[4] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," *ACM Queue*, vol. 9, no. 11, pp. 40–54, November 2011. [Online]. Available: http://doi.acm.org/10.1145/2063166.2071893

[5] D. Ros and M. Welzl, "Less-than-best-effort service: A survey of end-to-end approaches," *Communications Surveys Tutorials, IEEE*, In press. [Online]. Available: http://dx.doi.org/10.1109/SURV.2012.060912.00176

[6] D. A. Hayes and G. Armitage, "Revisiting TCP congestion control using delay gradients," in *IFIP NETWORKING 2011*, Valencia, Spain, May 2011, pp. 328–341.

[7] L.Stewart, D. Hayes, G. Armitage, M. Welzl, and A. Petlund, "Multimedia-unfriendly TCP congestion control and home gateway queue management,," in *ACM Multimedia Systems Conference (MMSys 2011)*, Feb 2011.

[8] M. Allman, V. Paxson, and E. Blanton, "TCP Congestion Control," RFC 5681, Internet Engineering Task Force, September 2009. [Online]. Available: http://tools.ietf.org/html/rfc5681

[9] J. Gettys, "The criminal mastermind: bufferbloat!" Dec 2010. [Online]. Available: http://gettys.wordpress.com/2010/12/03/introducing-the-criminal-mastermind-bufferbloat/

[10] T. Li, D. Leith, and D. Malone, "Buffer sizing for 802.11-based networks," *IEEE/ACM Transactions on Networking*, vol. 19, no. 1, pp. 156 –169, Feb 2011.

[11] S. Choi, K. Park, and C.-k. Kim, "On the performance characteristics of WLANs: revisited," *SIGMETRICS Perform. Eval. Rev.*, vol. 33, no. 1, pp. 97–108, Jun 2005.

[12] J. Choi, K. Park, and C. kwon Kim, "Cross-layer analysis of rate adaptation, dcf and tcp in multi-rate wlans," in *IEEE INFOCOM 2007*, May 2007, pp. 1055 –1063.

[13] N. Khademi, M. Welzl, and S. Gjessing, "Experimental evaluation of TCP performance in multi-rate 802.11 WLANs," in *2012 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, June 2012, pp. 1–9.

[14] N. Khademi, M. Welzl, and R. Lo Cigno, "On the uplink performance of TCP in multi-rate 802.11 WLANs," in *IFIP NETWORKING 2011*, Valencia, Spain, May 2011, pp. 368–378.

[15] R. T. Morris, J. C. Bicket, and J. C. Bicket, "Bit-rate selection in wireless networks," Master thesis, MIT, Tech. Rep., May 2005.

[16] "Minstrel description." [Online]. Available: http://madwifi-project.org/browser/madwifi/trunk/ath_rate/minstrel/minstrel.txt

[17] R. Jain, "A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks," *SIGCOMM Comput. Commun. Rev.*, vol. 19, no. 5, pp. 56–71, Oct 1989.

[18] Z. Wang and J. Crowcroft, "Eliminating periodic packet losses in the 4.3-Tahoe BSD TCP congestion control algorithm," *SIGCOMM Comput. Commun. Rev.*, vol. 22, no. 2, pp. 9–16, Apr 1992.

[19] L. S. Brakmo and L. L. Peterson, "TCP Vegas: end to end congestion avoidance on a global Internet," *IEEE J. Sel. Areas Commun.*, vol. 13, no. 8, pp. 1465–1480, Oct 1995.

[20] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: motivation, architecture, algorithms, performance," *IEEE/ACM Trans. Netw.*, vol. 14, no. 6, pp. 1246–1259, Dec 2006.

[21] R. King, R. Baraniuk, and R. Riedi, "TCP-Africa: An adaptive and fair rapid increase rule for scalable TCP," in *IEEE INFOCOM 2005*, Mar 2005, pp. 1838–1848.

[22] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound TCP approach for high-speed and long distance networks," in *IEEE INFOCOM 2006*, Apr 2006, pp. 1–12.

[23] D. A. Hayes and G. Armitage, "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.

[24] D. Leith, R. Shorten, G. McCullagh, L. Dunn, and F. Baker, "Making available base-RTT for use in congestion control applications," *IEEE Communications Letters*, vol. 12, no. 6, pp. 429 –431, Jun 2008.

[25] A. Venkataramani, R. Kokku, and M. Dahlin, "TCP Nice: a mechanism for background transfers," *SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 329–343, Dec. 2002.

[26] A. Kuzmanovic and E. Knightly, "TCP-LP: a distributed algorithm for low priority data transfer," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 3, 2003, pp. 1691–1701 vol.3.

[27] ——, "TCP-LP: low-priority service via end-point congestion control," *IEEE/ACM Trans. Netw.*, vol. 14, no. 4, pp. 739–752, Aug 2006.

[28] T. Tsugawa, G. Hasegawa, and M. Murata, "Background TCP data transfer with Inline network measurement," in *Communications, 2005 Asia-Pacific Conference on*, 2005, pp. 459–463.

[29] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)," RFC 6817, Internet Engineering Task Force, December 2012. [Online]. Available: http://tools.ietf.org/html/rfc6817

[30] D. Ros and M. Welzl, "Assessing LEDBAT's Delay Impact," *Communications Letters, IEEE*, 2013 (in press). [Online]. Available: http://dx.doi.org/10.1109/LCOMM.2013.040213.130137

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

[32] "NewTCP project tools," 2012. [Online]. Available: http://caia.swin.edu.au/urp/newtcp/tools.html

[33] "Emulab." [Online]. Available: http://www.emulab.net/

[34] M. Carbone and L. Rizzo, "Dummynet revisited," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 2, pp. 12–20, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1764873.1764876

[35] S. Zander, G. Armitage, L. M. Thuy Nguyen, and B. Tyo, "Minimally intrusive round trip time measurements using synthetic packet-pairs," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 060707A, Jul 2006. [Online]. Available: http://caia.swin.edu.au/reports/060707A/CAIA-TR-060707A.pdf

[36] "tcptrace." [Online]. Available: http://www.tcptrace.org/

[37] "tcpreplay." [Online]. Available: http://tcpreplay.synfin.net/