Enhancing TCP to Support Rate-Limited Traffic

Arjuna Sathiaseelan
School of Engineering
University of Aberdeen
Aberdeen, UK
arjuna@erg.abdn.ac.uk

Raffaello Secchi
School of Engineering
University of Aberdeen
Aberdeen, UK
raffaello@erg.abdn.ac.uk

Godred Fairhurst
School of Engineering
University of Aberdeen
Aberdeen, UK
gorry@erg.abdn.ac.uk

ABSTRACT
This paper introduces a new TCP congestion control mechanism for rate-limited applications that transmit data in bursts and do not fully utilise their allowed transmission rate. We propose "new-CWV", a method that allows a TCP connection to restart quickly from either an idle or application-limited period. Simulation results show that this provides faster convergence to the rate requested by a rate-limited application, demonstrated by a higher throughput, and better utilisation of unused capacity compared to Standard TCP or TCP with Congestion Window Validation.

Categories and Subject Descriptors
C.2.TCP/IP

Keywords
TCP, Congestion-control, capacity-sharing

1. INTRODUCTION
Many current Internet applications can be characterised as "rate-limited". We define a rate-limited application as one that transmits at a rate not directly controlled by the transport protocol, but instead dictated by the application. A rate-limited application may send at a Constant Bit Rate (CBR), less than limited by the transport, or send with periods of higher (but limited) rate, separated by periods with a much lower rate (application-limited periods), or by periods where no data is sent (idle periods).

Transmission Control Protocol (TCP), RFC 793 was designed to support a range of applications, but TCP congestion control [1] been optimised primarily for bulk transfers. The performance of bulk applications is limited by the TCP window. Bulk applications are not rate-limited.

Many multimedia applications use the User Datagram Protocol (UDP) specified in RFC 768. Such applications can (and often do) transmit at a constant rate, irrespective of available capacity and although applications may implement congestion control, they typically do not use a standard method.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CSWS'12, December 10, 2012, Nice, France.
Copyright 2012 ACM 978-1-4503-1780-1/12/12...$15.00.

It is commonly perceived that the Additive Increase Multiplicative Decrease (AIMD) behaviour of TCP is inappropriate for the strict requirements for timeliness of interactive applications. This led to initiatives to define alternate methods such as the TCP Friendly Rate Control (RFC 5348), although at present this has yet to achieve wide scale deployment [2]. The recent growth of TCP-based multimedia applications has reopened the debate on use of TCP for rate-limited applications (e.g.[3],[4]). Other rate-limited TCP behaviours include HTTP 1.1 persistent connections, Google SPDY (which uses persistent connections to retrieve multiple objects) and HTTP Adaptive Streaming (HAS).

This paper will focus on such traffic, for which Standard TCP is not generally well-adapted. One method to provide acceptable performance over TCP is for the sender to continue to transmit at the same rate during periods of silence, known as "padding". Although this can ensure acceptable application performance, it can also degrade network performance and decrease the effectiveness of TCP congestion control [2][5].

The paper proposes a set of modifications to Standard TCP to enable effective use of standards-based methods. Our proposal, new-CWV, freezes the congestion window (*cwnd)* during a rate-limited period, enabling the application to restart with the same *cwnd* after a rate-limited period. Hence, the *cwnd* would neither grow nor reduce while rate-limited. This allows applications to more quickly resume transmission. When a new-CWV sender detects a congestion event during the first RTT of restarting with a large *cwnd*, the methods appropriately reduce the *cwnd*. This can satisfy the capacity requirements of real-time applications and at the same time provide congestion control appropriate for use in the Internet. We suggest this may encourage more application developers to use standards-based TCP congestion control.

A similar notion of freezing the *cwnd* was also proposed in Freeze-TCP [6], however Freeze-TCP was specifically proposed to mitigate mobility related disconnections and is not a suitable solution for variable-rate congestion control.

The remainder of the paper is organized as follows: Section 2 discusses the behaviour of TCP when carrying rate-limited traffic. Section 3 describes new-CWV. Section 4 analyses the performance of new-CWV over a range of scenarios. This is followed by the conclusions.

1. RATE-LIMITED TRAFFIC OVER TCP

Standard TCP uses the *cwnd* to limit the number of bytes/packets a TCP flow may have in a network path at any time. The *cwnd* starts at a value known as the Initial Window (IW). This is updated by the AIMD algorithm, as TCP continuously probes for additional capacity. TCP also maintains a variable, the flight size, *w_used*, that reflects the volume of unacknowledged data. This is always less than *cwnd*. The algorithm maintains a variable, called the Slow-Start threshold (*ssthresh*), which reflects the available path capacity at the time of the last congestion event.

Experience shows that the AIMD algorithm performs well for bulk applications such as FTP, where continuous data is available at the sender, limiting transmission to an average rate that is of the order of the fair share of the capacity along the path. Instead this paper focuses on the behaviour of TCP when carrying rate-limited traffic characterised by periods of higher (but limited) transmission rate, separated by periods in which much less (application-limited period), or no data is sent (idle period). In terms of TCP, these applications do not consume the entire *cwnd*.

Standard TCP dictates that when an application is idle for a period greater than the Retransmission Timeout (RTO), the *cwnd* is reset to no more than the Restart Window (RW) [1]. During an application-limited period, a Standard TCP sender continues to grow the *cwnd* for every received acknowledgement (ACK), allowing the *cwnd* to reach an arbitrarily large value. However, when the packet probes along the transmission path are sent at a lower rate than permitted by *cwnd*, the reception of an ACK does not provide evidence that the network path was able to sustain the transmission rate reflected by *cwnd*. The result is an 'invalid' *cwnd*, i.e. *cwnd* increasingly becomes a poor estimate of the available path capacity. If an application with an invalid *cwnd* were to suddenly increase its transmission rate, the sender would be allowed to immediately inject a significant volume of additional traffic into the network. This could lead to severe congestion. It is also suggested to be a substantial performance issue for many rate-limited applications and has been a dis-incentive for such application designers to consider TCP [7] [8].

TCP Congestion Window Validation (TCP-CWV) [9] was proposed as an experimental standard by the IETF in RFC 2861. It was seen as a remedy for some of the problems imposed by TCP carrying a rate-limited application. TCP-CWV modified the use of *cwnd* during an idle or application-limited period: During an idle period greater than one RTO, *cwnd* is reduced by a half each time the connection has been idle for an RTO period. This is equivalent to exponentially decaying *cwnd* during the idle period. TCP-CWV modified the standard congestion control algorithm during an application-limited period, when the *cwnd* had not been fully utilised for a period larger than an RTO. It recommended that *cwnd* is reduced to (*cwnd*+*w_used*)/2 for each packet transmission that does not utilise the full *cwnd* with an empty transmission buffer for more than RTO seconds, where *w_used* is the estimated used portion of *cwnd*. This avoids growth of *cwnd* to a value larger than the capacity utilised by an application.

Our previous work showed that neither TCP nor TCP-CWV were suitable for rate-limited applications [8][10]. During an idle period longer than one RTO, Standard TCP reduced the *cwnd* to the RW and then slow-started to the application rate. This approach may be acceptable to the network, but does not benefit the application. TCP-CWV mitigates this and benefits the application, allowing it to send packets faster after idle. However, in the presence of a long idle period (several RTO periods) TCP-CWV would reduce the *cwnd* to the RW and perform similar to Standard TCP. During an application-limited period, TCP-CWV performance can be lower than Standard TCP. For an application-limited period longer than a RTO, Standard TCP sends more aggressively than TCP-CWV. Standard TCP benefited application-limited traffic (as shown in Fig. 3), whereas TCP-CWV is more conservative.

It is therefore not clear what to recommend to an application designer wishing to support rate-limited traffic. TCP-CWV is of benefit if an application exhibits regular idleness and the idle period is less than a few RTOs. Applications exhibiting large idle periods (e.g. tens of seconds) would get no benefit or loss when using TCP-CWV compared to Standard TCP. Moreover, the conservative approach of TCP-CWV during application-limited periods is a disincentive to rate-limited applications, resulting in a low initial restart rate. This explains why TCP-CWV has not attracted more use – it was inconsistent and the incentives offered did not suit rate-limited applications with a mixture of idle and application-limited periods.

3. new-CWV

This section presents new-CWV [11], a set of proposed modifications to Standard TCP designed to mitigate the problems imposed by both Standard TCP and TCP-CWV. We suggest the transport should not try to identify idle periods, but instead be driven by the actual traffic sustained by the network path. new-CWV does not therefore differentiate between idle and application-limited periods.

The new method uses Standard TCP with the SACK option [12]. It updates TCP by freezing the *cwnd* during rate-limited periods. This allows an application to later resume at the same rate before the application became rate-limited. The sender uses a new variable *pipeACK*, the actual volume of data that was acknowledged by the network per measured RTT. When this is less than half the *cwnd*, new-CWV enters a phase where the transmission rate is no longer constrained by the *cwnd*. This is called the non-validated period (NVP). During this period, the *cwnd* neither grows nor reduces. This phase concludes after a fixed period of time (5 minutes, as explained below) or when the sender transmits sufficient data so that *pipeACK*

is greater than half *cwnd* (i.e. it is no longer rate-limited) or when there is a congestion signal (as detailed below).

One reason for selecting a 5 minute period is that during the absence of an application-specified user timeout, the TCP specification defines a default user timeout of 5 minutes i.e. how long transmitted data may remain unacknowledged before a connection is forcefully closed. The value of 5 minutes should be seen as a compromise, sufficient for most applications. For most practical applications, the performance is not significantly different to that observed using a non-standard method that does not reset *cwnd* after a rate-limited period, but avoids the undesirable side effects that can result if *cwnd* is preserved for an arbitrary period, which was a part of the problem that TCP-CWV originally attempted to address. [12] provides more discussions on why we chose a 5 minute period.

new-CWV assumes that, if a sender is rate-limited (either idle or application-limited) for more than the NVP, then *cwnd* is no longer an acceptable estimate of the path capacity. It therefore reduces the *cwnd* to max(1/2**cwnd*, IW). If *ssthresh* was low compared to *cwnd,* it also increases *ssthresh* to max(*ssthresh*, 3**cwnd*/4). The decision to update *ssthresh* is because TCP has successfully sustained the current rate, even if this is higher than the previous *ssthresh* value. Hence, it is safe to increase a previously reduced *ssthresh*, allowing more rapid convergence to the previous *cwnd*. The weighting three-fourths is to avoid excessive overshoot, as noted in [13].

There are potential risks in injecting too large a burst of packets during a restart phase [13]. In this phase, TCP is not receiving ACKs confirming that packets have left the network, and could generate sequence of back-to-back packets that cause significant packet loss at a bottleneck. This problem may also occur when an application transmits a large block of data during the NVP, even though a sender may be rate-limited. A suitable remedy may be to enable TCP packet pacing, or to limit burst size [11]. Since these are already deployed TCP mechanisms, various techniques are analysed in [13], and are not discussed further here.

During the NVP, new-CWV seeks to determine whether the currently used rate is still safe for the Internet path, i.e. that the restarting sender did not induce congestion: If the RTO expires during the NVP, the sender uses the Standard TCP mechanism. This resets *cwnd* to RW and new-CWV exits the NVP. *pipeACK* should also be reset. If the sender receives congestion feedback while in the NVP, i.e. it detects a packet-drop or receives an Explicit Congestion Notification (ECN), the sender must reduce the *cwnd*. These events indicate that it was unsafe to start with the higher *cwnd*, and TCP must quickly reduce the rate to avoid further congestion of the network path.

Following detected congestion, new-CWV attempts to estimate a new safe *cwnd* by estimating a fair share of the path (also known as "the pipe"). It calculates this using received SACK information, as described [12]. The flight size, *w_used* (D) is reduced by the number of packets detected as lost in the SACK information (R). At the end of the recovery phase, new-CWV resets the *cwnd* to half the present *w_used*, rather than a half of the stored *cwnd*. Although this requires SACK to be enabled, it selects a *cwnd* based on the measured path capacity, better reflecting the fair-share. A similar approach was proposed by TCP Jump Start [14], as a congestion response after more rapid opening of a connection.

In the following section, we consider two other alternative mechanisms to reduce the *cwnd* following loss during the NVP: one resets cwnd to the RW. The other uses the standard Fast Retransmit/Fast Recovery algorithm. We found that our proposed reduction method where the *cwnd* is reduced to half the present *w_used* (known as (D-R)/2 [14]) is the better behavior.

The introduction of NVP improves the performance of an application that exhibits frequent rate-limited periods. The benefit of new-CWV is greater for an application that would otherwise require padding. By removing the incentive for padding an application injects less data into the network, such an application that pauses or reduces its transmission rate for less than the NVP is not penalised when it resumes sending data. Resetting *ssthresh* at conclusion of NVP allows a more rapid (exponential) growth towards the previous *cwnd* should the application later restart at a higher rate.

4. SIMULATION ANALYSIS

We analysed a range of methods using the network simulator (ns-2) [15]. We considered a single bottleneck topology (Fig. 1) with two routers (n1-n2) and two access nodes (n0-n3). The access nodes were connected to the routers using a 100 Mb/s link (n1-n2) with a delay selected for each simulation scenario. The bottleneck path had a capacity of 100 Mbps and a one-way delay of 100 ms and a queue size set to the bandwidth-delay product.

The simulations considered rate-limited traffic sources with a rate of 512 kb/s, similar to medium quality interactive video [16]. These traffic sources could either be stopped and started for a period of time (simulating an idle period) or could have a lower rate (simulating an application-limited period). All flows randomly started based on a uniform distribution over a period of 1 s. TCP SACK was enabled. The Nagle and delayed ACK algorithms were disabled. The minimum RTO was 1 s.

We first analyze the dynamics of new-CWV to compare its capacity sharing with TCP and TCP-CWV. Two cases were considered: the case when a rate-limited application restarted after being idle for a few seconds, and the case when it restarted after being application-limited. We also compared the performance benefits of new-CWV with several other network characteristics [10].

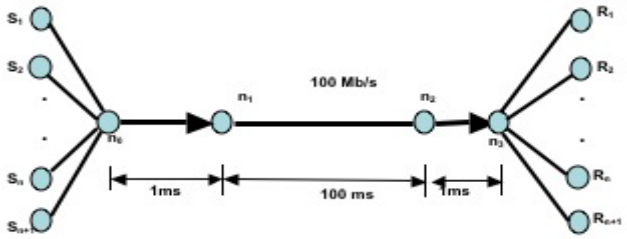

Figure 1. Bottleneck simulation topology

Figures 2 and 3 plot the sequence number dynamics for Standard TCP, TCP-CWV and new-CWV for a congestion-free scenario, showing the behaviour with a rate-limited application. There is improvement using new-CWV.

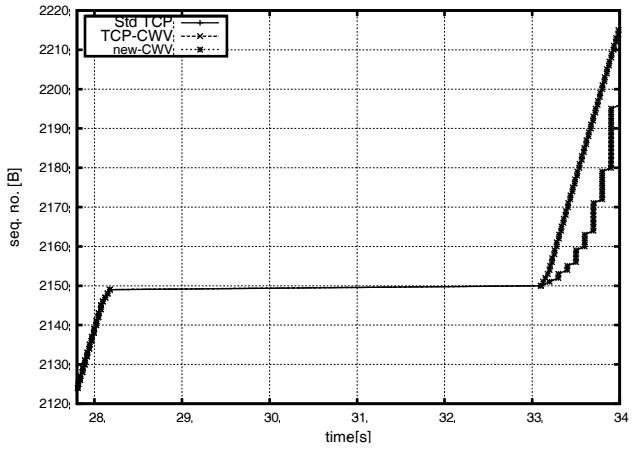

Figure 2. Idle Scenario: Sequence number dynamics for a congestion-free path using Standard-TCP, TCP-CWV and new-CWV.

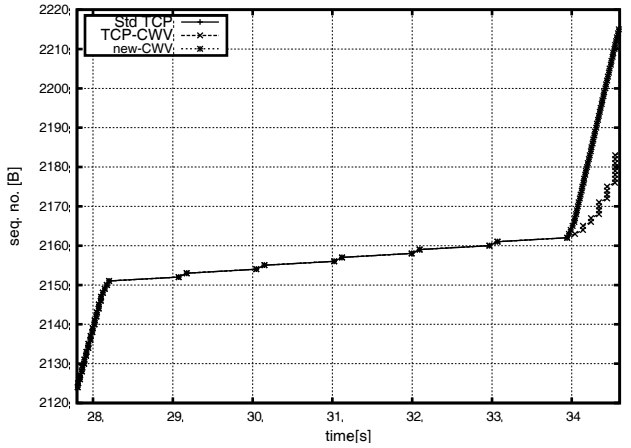

Figure 3. Application-limited Scenario: Sequence number dynamics for a congestion-free path using Standard-TCP, TCP-CWV and new-CWV.

A long idle period with TCP-CWV or Standard TCP triggered Slow-Start to increase *cwnd* from the RW that requires several round-trips during which applications cannot send at their desired rate. In an application-limited period TCP-CWV reduced *cwnd*. If the application then rapidly increased the sending-rate to the previous rate, TCP-CWV required several round-trips to restore the transmission rate. In the same conditions, new-CWV did not reduce *cwnd* during the rate-limited period (both idle and application-limited) and could therefore promptly resume with the sending-rate of the application.

The strategy of preserving *cwnd* for rate-limited periods much longer than allowed by Standard TCP is clearly effective for uncongested scenarios. However, a transient change in network state while the sender is rate-limited could result in a significant reduction in the fair rate. For example, in a wireless network, where there can be sudden changes following network hand-over, propagation condition changes. In wired networks, there can be route changes or the arrival of competing cross-traffic/flash crowds that reduce the available capacity. When the path changes, TCP needs to rapidly converge to reflect the new path characteristics. The following tests seek to evaluate the impact on network performance following a significant reduction in path capacity due to a transient condition while the sender was rate-limited.

A group of 512 kb/s flows were simulated. Each flow started at a random time uniformly distributed over 1 s. After 28 s, the flows stopped for 5 s. During this period, the bottleneck path reduced from 100 Mb/s to 2 Mb/s. After 5 s, the flows re-started to the original rate, at a time randomly distributed over a period of 1 s. Simulation results show the performance achieved by TCP, TCP-CWV and new-CWV and for an application that used padding. The protocol aggressiveness was evaluated by measuring the average arrival rate at the receiver (Figure 4). In the presence of congestion, the new-CWV variants exhibited better performance compared to both Standard TCP and TCP-CWV. Although this behaviour could be thought of as aggressive, it can be seen that average received rate achieved by the new-CWV flows was only about 3% higher than the TCP fair share (Figure 4).

When there was heavy congestion (from 16 flows), the average receive rate of all the new-CWV flows was less than or equal to the TCP Fair share (less than 0.1% difference). The experiment also measured the average packet drop rate at the bottleneck router over 10 RTTs (Figure 5). This illustrates a drawback of padding: Padding during idle periods led to high levels of packet drop under transient network conditions, even for a small number of connections sharing the path capacity.

One potential solution would be to reduce the effect of sending dummy packets to send only one dummy packet every RTT of idle period. However, this behaviour is similar to an application-limited behaviour and hence TCP would increase the *cwnd* by one segment for every acknowledged dummy packet. In periods of heavy congestion, sending even a single dummy packet could potentially worsen the congestion situation, and anyway

yields little information about actual path capacity. A new-CWV sender reacted to a change in path state while idle (i.e. within the NVP), by quickly backing-off after the first round-trip time. This reduced the average drop rate at the bottleneck router.

Figures 6 and 7 show results for an application-limited case: The sending rate was lowered from 512 kb/s to 12 kb/s for a 5 second period, during which the network capacity changed. Standard TCP allowed *cwnd* to unnecessarily grow and resulting in many packet drops during the first few RTTs (Figure 7). TCP-CWV mitigated this by proactively reducing the *cwnd* during the application-limited period, but achieving a throughput that was conservatively smaller than its fair-share of the path capacity (Figure 6).

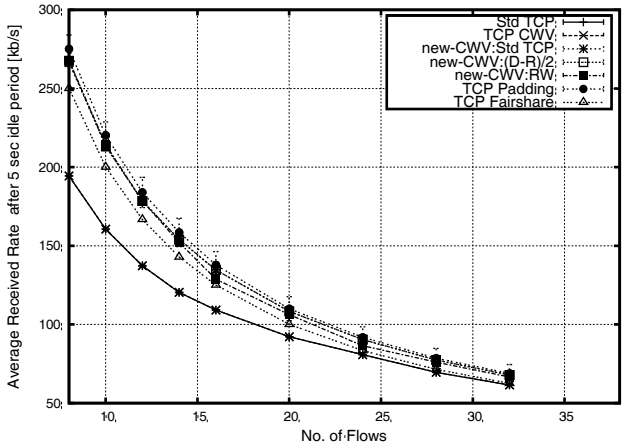

Figure 4. Average received rate for a set of flows after a 5s idle period.

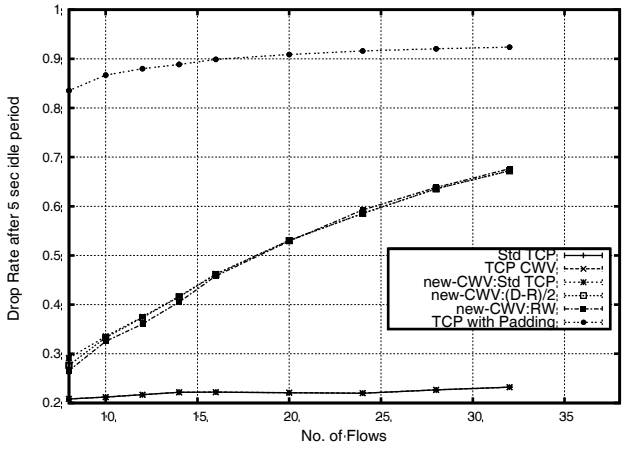

Figure 5. Bottleneck router packet-drop rate for a set of flows with a 5s idle period.

new-CWV presents a trade-off between these two extremes. Maintaining the most recent *cwnd*, it was able to offer a higher average receive rate than TCP-CWV (Figure 6) while inducing fewer packet drops than Standard TCP (Figure 7). These experiments did not reveal a significant difference in application or network performance between variants (less than 0.5% for both idle and application-limited cases).

Standard TCP reduced *cwnd* to a half after packet loss when new-CWV used the Standard TCP recovery. When new-CWV reduced *cwnd* to the RW, this reduced the average receiver rate. The best performance was observed when new-CWV reset *cwnd* to *w_used* less the volume of lost data, resembling the method used by Jump Start (where the new *cwnd* labelled "(D-R)/2"). In this case, the average received rate was similar to that of Standard TCP. We suggest this shows that it may be safe to switch to a higher sending rate at restart, providing that the sender also quickly reduces the rate on detecting congestion.

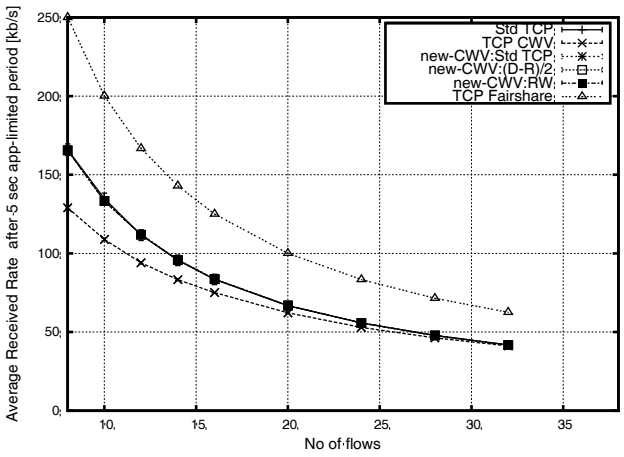

Figure 6. Average received rate for a set of flows after a 5s application-limited period.

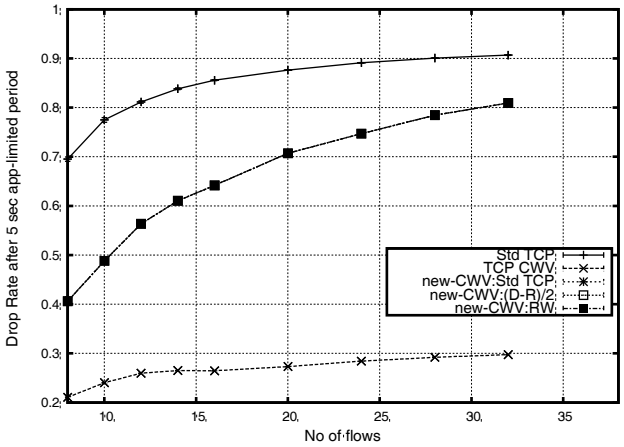

Figure 7. Bottleneck router packet drop rate for a set of flows after a 5s application-limited period.

These simulated results represent a pathological scenario when a single transient event led to significant reduction of bottleneck capacity. The performance in terms of the application and network are approximately proportional to such a change. For example, performance of all variants of new-CWV improved when the bottleneck capacity was changed to more than 2Mb/s. Results considering a wider range of scenarios can be found in [10]. There is a

drawback when flows use a larger *cwnd* during transient conditions, which is also a disadvantage for new-CWV. These flows can send more than Standard TCP for the first few RTTs following an idle or application-limited period, since they restart with a larger *cwnd*.

Standard TCP was only aggressive during an application-limited period compared to TCP-CWV (e.g. Figure 6). TCP-CWV was only aggressive compared to Standard TCP following an idle period of a few RTTs. However, if a flow experienced congestion, new-CWV quickly adapted to the bottleneck capacity, the loss therefore decreased and stabilised over a time, similar to both TCP and TCP-CWV. We also examined new-CWV behaviour for a range of path RTTs and found the "Received rate" was a linear function of path RTT [10]. However, we also found that reducing to the RW reduces the average received rate for a longer path RTT (i.e. > 100ms) and all variants converged for a large path RTT (e.g. 800 ms) because of retransmission timeout, which resulted in all variants being reset to the RW.

Although our paper analysed rate-limited traffic, representing interactive video, we believe that the new-CWV method offers benefit to a wide range of rate-limited traffic. This includes applications such as variable-rate video, and persistent web-based applications, such as HTTP 1.1, Google SPDY and HTTP Adaptive Streaming. In summary, our analysis suggests that new-CWV provides an acceptable congestion response for supporting rate-limited traffic in the Internet. This method is currently being proposed for consideration by the IETF [11].

5. CONCLUSION

Past work has shown that both TCP and TCP-CWV where not suited to rate-limited applications that did not fully consume the TCP *cwnd*. Hence there is a need to update TCP to effectively support rate-limited traffic.

This paper proposes a method called new-CWV that updates Standard TCP to offer the required performance improvement. Using the new method, a rate-limited application experiences a faster response than TCP-CWV, since it does not reduce the *cwnd* during rate-limited periods. new-CWV also adds the concept of a non-validated period to provide robustness to changes in the characteristics of the network path. Simulations compared the response for different methods after rate-limited periods. new-CWV has the best application performance of the evaluated methods and provides a faster response to transient congestion. We also showed that new-CWV is normally able to recover from the congestion quicker and by inducing fewer packet drops than Standard TCP would have when an application used padding during rate-limited periods. Standardisation of the new method would offer incentive for application designers to use a standard congestion-controlled transport for rate-limited traffic, rather than resorting to non-congestion controlled protocols or expedient application-layer methods.

6. ACKNOWLEDGEMENTS

Dr. Sathiaseelan was supported by the RCUK Digital Economy programme to the dot.rural Digital Economy Hub; award reference: EP/G066051/1. The authors gratefully acknowledge the analysis and detailed simulation work performed by Dr Israfil Biswas for his PhD studies.

REFERENCES

[1] M. Allman, V. Paxson, E. Blanton, TCP Congestion Control, IETF RFC 5681, Sept. 2009.

[2] A. Sathiaseelan, G. Fairhurst, TCP Friendly Rate Control (TFRC) for Bursty Media Flows, 34(15), Sept. 2011, pp. 1836-1847.

[3] S. Baset, E. Brosh, V. Misra, D. Rubenstein, H. Schulzrinne, Understanding the behaviour of TCP for real-time CBR workloads, ACM CoNEXT, Lisbon, Portugal, Dec. 2006.

[4] E. Brosh, S. Baset, D. Rubenstein, H. Schulzrinne, The Delay-Friendliness of TCP, ACM SIGMETRICS, Annapolis, MD, USA, Jun. 2008.

[5] E. Kohler, S. Floyd, A. Sathiaseelan, Faster Restart for TCP Friendly Rate Control (TFRC), IETF Work in Progress (expired), Jul. 2008.

[6] T. Goff, J. Moronski, D. S. Phatak, and V. Gupta, Freeze-TCP: a True End-to-end TCP Enhancement Mechanism for Mobile Environments, IEEE INFOCOM, Tel-Aviv, Israel, 2000, pp. 1537-1545.

[7] S. Hughes, J. Touch, J. Heidemann, Issues in TCP Slow-Start Restart After Idle, IETF Work in Progress (expired), Dec. 2001.

[8] I. Biswas, A. Sathiaseelan, R. Secchi, G. Fairhurst, Analysing TCP for Bursty Traffic, Int'l J. of Comm, Network and System Sciences, 7(3), July 2010.

[9] M. Handley, J. Padhye, S. Floyd, TCP Congestion Window Validation, IETF RFC 2861, Jun. 2000.

[10] I.Biswas, Internet congestion control for variable rate TCP traffic, PhD Thesis, University of Aberdeen, Aberdeen, UK, 2011.

[11] G. Fairhurst, A.Sathiaseelan, "Updating TCP to support Variable-Rate Traffic", IETF Work in Progress, Sept. 2012.

[12] E. Blanton, M. Allman, K. Fall, L. Wang, A Conservative SACK-based Loss Recovery Algorithm for TCP, IETF RFC 3517 Apr. 2003.

[13] M. Allman, E. Blanton, Notes on Burst Mitigation for Transport Protocols, ACM CCR, 35(2), Apr. 2005.

[14] D. Liu, M. Allman, S. Jiny, L. Wang, Congestion Control without a Startup Phase, 5th International PFLDnet, Los Angeles, California, USA, Feb. 2007.

[15] S. McCanne, S. Floyd, *Network Simulator*, http://www.isi.edu/nsnam/ns/

[16] S. Floyd, M. Handley, E. Kohler, Problem Statement for the DCCP, IETF RFC 4336, Mar. 2006.