Performance Evaluation of Congestion Window Validation for DASH Transport

Sajid Nazir
School of Engineering
University of Aberdeen
AB24 3UE, Aberdeen, UK
sajid.nazir@abdn.ac.uk

Ziaul Hossain
School of Engineering
University of Aberdeen
AB24 3UE, Aberdeen, UK
ziaul.hossain@abdn.ac.uk

Raffaello Secchi
School of Engineering
University of Aberdeen
AB24 3UE, Aberdeen, UK
raffaello@erg.abdn.ac.uk

Matthew Broadbent
School of Computing and Communications
Lancaster University
LA1 4WA, Lancaster, UK
m.broadbent@lancaster.ac.uk

Andreas Petlund
Department of Informatics
Postboks 1080 Blindern
University of Oslo
0316 Oslo, Norway
apetlund@ifi.uio.no

Gorry Fairhurst
School of Engineering
University of Aberdeen
AB24 3UE, Aberdeen, UK
gorry@erg.abdn.ac.uk

ABSTRACT

A recent proposed update to TCP congestion control, TCP-newCWV, has targeted congestion control for rate-limited applications. These methods need to be explored in the context of rate-adaptive applications, such as DASH. The new method enables a client to exploit the persistence of a DASH connection and enables the DASH server to rapidly resume transmission of a series of video segments using a single TCP connection. Another technique, called 'Pacing' smoothes DASH bursts when there is no TCP ACK clock, and is shown to significantly reduce burst loss. These two methods in combination can increase the application performance. This paper investigates the effect of implementing these techniques on a DASH flow in different congestion scenarios and whether the method can promote better capacity sharing while minimizing the latency experienced by other flows sharing a common network bottleneck. The results confirm that newCWV with Pacing provides a benefit as a platform for DASH transport.

Categories and Subject Descriptors

C.2.2 [**Computer-communication Networks**]: Network protocols-*Protocol verification*

General Terms

Experimentation, measurement, performance

Keywords

Congestion window, adaptive streaming, DASH, network capacity sharing, pacing.

NOSSDAV '14, March 19 - 21 2014, Singapore, Singapore
Copyright 2014 ACM 978-1-4503-2706-0/14/03…$15.00.
http://dx.doi.org/10.1145/2578260.2578275

1. INTRODUCTION

The volume of video traffic using the Internet continues to grow, with video expected to dominate network traffic in the near future [1]. In video transmission there has been a shift from traditional RTP/UDP based streaming to HTTP/TCP based streaming, as evidenced by related adaptive streaming solutions from Adobe, Apple and Microsoft. This migration to HTTP streaming is favored for many reasons [2].

Recently, MPEG-DASH has been standardized [3]. This targets diverse devices such as smartphones, tablets, TV set-top boxes, Internet TV and computers to offer multimedia content across a range of network capacity. It defines a Media Presentation Description (MPD) and video segments, but does not define client behavior nor the encoder.

A DASH client requests video segments using the MPD information. Each segment is sent using TCP and comprises a series of IP packets. Many clients open a separate TCP connection to request each video segment. This process continues until the client buffer is filled, when playback starts. Thereafter, video segments are only requested once the buffer drains, typically waiting until it has reached one-third of full capacity. This cycle repeats creating traffic with an ON state (during which a connection downloads data), and an OFF state (when the connection is idle).

DASH also permits a persistent mode, in which a single TCP connection is used to transfer multiple video segments. This alternating active and idle transmission pattern can interact poorly with standard TCP transport, causing the congestion window (*cwnd*) [4] to shrink after every idle period and restart each time using slow start. This reduces the ability of the DASH server to utilise available network capacity [5].

DASH quality may be improved by modifying the client or server behavior, and this has been an active area of research. The difficulty of client-side estimation of network capacity is highlighted in [6], where rate selection based on wrong estimates lead to variable and low-quality video. The selected video services [6] were shown to underestimate network capacity due to interactions between HTTP and TCP congestion control. The bursts of traffic from HTTP adaptive streaming can also adversely impact other network traffic [7] and a client-side scheme has been proposed to mitigate queuing in a network bottleneck. Instability of competing adaptive streaming players sharing a bottleneck has also been demonstrated [8], where traffic shaping mechanism at the server has been proposed.

This paper therefore evaluates the performance of the newCWV update to TCP [9] when providing transport to a DASH service. We show that when DASH uses a combination of newCWV and Pacing at the transport layer [10], it is more network-friendly and reduced the impact on other sessions that share the bottleneck. This can also improve the performance of DASH, by eliminating the need for slow start when transmitting each message.

The remainder of this paper is structured as follows: Section 2 describes the relevant features of DASH and TCP. The experimental setup is explained in Section 3. The results are provided in Sections 4 and 5. Finally, conclusions and future work are given in Section 6.

2. BACKGROUND

DASH operates above the TCP transport service. A DASH client may decide whether to set-up a new TCP connection (3-way TCP handshake) or re-use an already existing connection while requesting a video segment.

2.1 DASH Persistence

A non-persistent TCP connection uses slow-start to probe for capacity at the start of each individual video segment. A DASH session that can reuse a previous TCP connection to deliver new video segments is termed *persistent*. Persistence requires that a server maintains the connection state for each client and has the disadvantage that it could result in open but unused connections if a client pauses or silently leaves a session.

Persistence also has merits. A DASH client can use persistence to avoid the overhead of opening a new connection [11], and in so-doing can build better picture of the network capacity available for download.

While the current TCP standard supports applications that use persistent connections, such as DASH, it requires each persistent connection to Slow Start after an idle period [4] from the TCP Initial Window (IW). This network behavior resembles that of a series of separate connections when persistence is not used. It can improve traffic sharing a congested network path.

2.2 Cross-layer Interaction

TCP and HTTP have different objectives and approaches for adapting video segment download to available network capacity.

On the one hand, TCP at the Transport Layer primarily seeks to act as a good network citizen by reducing its transmission rate in the face of impending congestion, and (slowly) probing for new capacity each time the application needs to send more than it has sent previously. This sensing of capacity occurs each time the TCP sender receives an ACK, i.e. each network path Round Trip Time (RTT).

On the other hand, a DASH client works at the Application Layer seeking to choose a download video segment size that optimizes the quality of the video within a time bounded by the receiver buffer size. The clients typically adjust the download by measuring the download time of each segment, responding much more slowly than TCP – but with the advantage that it can actually adjust the size of each segment based on current conditions. These two feedback loops interact with each other to determine the traffic characteristics.

2.3 TCP newCWV

The Standard TCP behavior is to collapse the *cwnd* for connections that have been idle for a period longer than TCP Retransmission Time Out (RTO). Standard TCP also continues to grow the *cwnd* every time an ACK is received for rate limited connections. As the *cwnd* grows, it no longer relates to the application sending rate and becomes "invalid".

Congestion Window Validation, CWV [12] was an experimental sender-side method proposed by the IETF to regulate the rate of TCP applications that send bursts of data. It proposed to restrict the *cwnd* from unnecessary growth or collapse depending on the current sending rate. It used the FlightSize, i.e. the amount of outstanding data not yet acknowledged, to determine if *cwnd* was valid. The FlightSize reflects the utilized path capacity at the moment a loss is detected; but it does not reflect the path capacity during normal transfer if the application is rate-limited. However, experience showed that CWV reduced the *cwnd* too conservatively for many rate-limited applications [5]. CWV has seen limited deployment in Linux, but is often not used by interactive bursty applications.

A growth in the use of bursty network applications has renewed interest in congestion window validation. It has been shown the motives of CWV were good, but that the proposed experimental method had a number of issues. newCWV has therefore been proposed at the IETF [9] to address the shortcomings in CWV and permit a more efficient use of the available capacity for rate-limited applications.

The new method does not attempt to differentiate idle from rate-limited application behavior and provides a new way to determine if an application is rate-limited. This introduces *pipeACK* as a measure of the amount of acknowledged data recently exchanged over the network path. It also specifies the Validated phase, which is when the following condition holds for a TCP sender:

$$pipeACK = \frac{1}{2} * cwnd \qquad (1)$$

In newCWV, the *cwnd* is allowed to grow only if either the sender is in the validated phase or the *cwnd* has been fully utilised. This results in *cwnd* being frozen when a sender has sent less than half the current *cwnd*. The *cwnd* is kept at this value for the duration of the non-validated period, after which it is halved to prevent senders preserving a large unused value indefinitely.

2.4 TCP Pacing

Since the DASH service transfers video segments greater than a single network packet, it can potentially result in bursts of network traffic after each request. These burst of activity can adversely impact any flow that shares a common bottleneck with the DASH flow, inducing queuing latency and increasing the probability of packet loss.

In normal (bulk) transmission the TCP sender rate is controlled by the arrival of TCP ACKs, in effect each arriving ACK identifies that a packet has been successfully sent along the network path, and hence triggers transmission of new data (known as “ACK Clocking”). This clocking effect does not occur in the non validated phase, since the sender is not cwnd-limited. Hence, newCWV modifies TCP sender behavior in the non-validated phase to control the maximum traffic burst allowed to be sent to the network.

Spreading the packet transmissions over a period of time is usually referred to as "Pacing" [10]. The most appropriate method to implement Pacing depends on the design of the TCP/IP stack, speed of interface and whether hardware support is used (such as TCP Segmentation Offload, TSO [13]).

Limiting the burstiness of a flow using Pacing results in smaller network queues (less congestion), reduces buffering requirements and also reduces latency experienced by flows [10, 14]. We show that Pacing with newCWV reduces significantly packet losses when used with DASH and that this improves capacity sharing and therefore the performance of other flows. However, some authors have indicated potential instability problems if Pacing were to be widely deployed [15, 16].

3. EXPERIMENTAL SETUP

The common configurations for the set of experiments are described in this Section, including the test bed and traffic generating tools for cross traffic.

Clients and servers used the Linux operating system (kernel version >3.0). The web server was installed with the standard Apache web server [17]. For traffic control, the Linux traffic netem [18] tool was used to form a bottleneck router to emulate a 1 Mbps link with a propagation delay of 200 ms. A token bucket filter set the rate of the router with a drop-tail policy and a queue size of 30KB.

The experiments used a default configuration of TCP New Reno. This was compared with a TCP New Reno implementing the newCWV kernel module. During the experiments with Pacing, pacing was enabled in conjunction with newCWV. Several different mechanisms can be used to implement Pacing. We used the version of TCP Pacing that was included in Linux kernel 3.12, enabled by using Fair Queuing scheduling [19]. Our test bed verification and [23] confirm that the applied technique actually spreads out the bursts over a period of RTT.

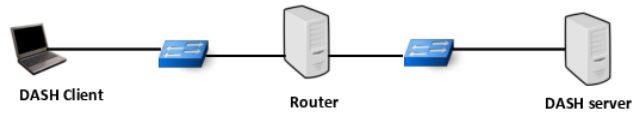

Figure 1. Experimental setup for a single flow.

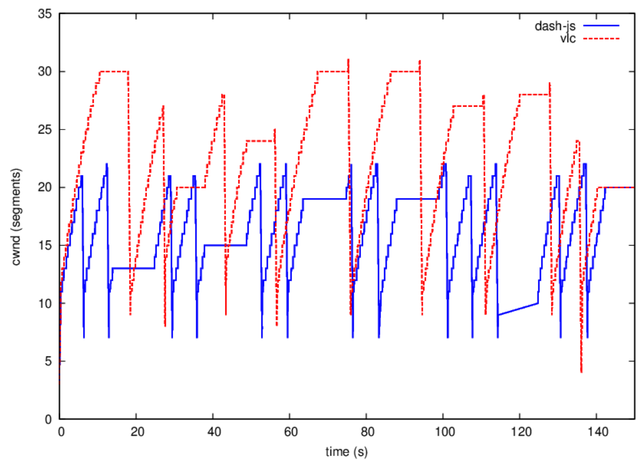

Figure 2. *cwnd* with newCWV for DASH-JS and VLC; DASH-JS encounters more losses compared to VLC player.

3.1 Video Content

The DASH web server hosted a video dataset using video from *Big Buck Bunny* [20] at a video resolution of 480x360, where each video segment was 2 seconds. The encoded bit rates at this resolution were 200, 250, 300, 400, 500, 600 and 700kbps. This DASH dataset was used for all experiments.

3.2 DASH Client

The DASH client used DASH-JS [21] running with a Google Chrome and a VideoLAN Clinet (VLC) media player [22] client. DASH-JS maintains a 30 seconds buffer for the video segments.

The client download algorithm was as follows: Initial video segments were continuously requested until the buffer was full. Each time the buffer emptied by one-third, more video segments were requested to refill the buffer.

The VLC player implements a DASH client with persistent behavior. Since VLC version 2.1.0 did not play back the video smoothly, we modified its buffer policy to match that of DASH-JS [21]. Both DASH-JS and VLC were used for single flow experiments, whereas only VLC was used for the scenarios with multiple flows.

4. TRAFFIC CHARACTERIZATION WITH A SINGLE DASH FLOW

This section presents experiments with a single DASH client to characterize the behavior of using newCWV. The results are provided for both the DASH-JS and the VLC clients. The experiments used the test bed depicted in Figure 1 with a bottleneck of 1 Mbps.

Both clients enabled persistent downloads and it was observed that this significantly reduced the number of TCP sessions used. In total, the DASH-JS client opened only eight to twelve connections in series to the DASH server. The VLC client used four persistent TCP connections. Both strategies for opening connections are consistent with the DASH specifications and provide examples of traffic with different characteristics.

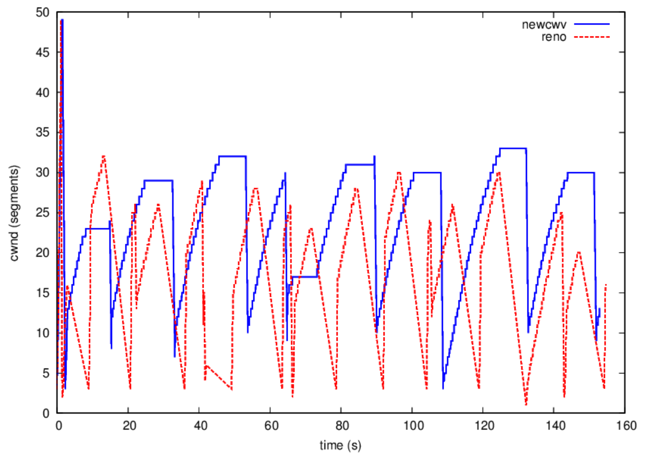

Figure 3. *cwnd* for VLC with newCWV and TCP NewReno; with newCWV (blue solid line) the *cwnd* is frozen during idle periods (horizontal lines). Using TCP NewReno the *cwnd* was reset after each idle period.

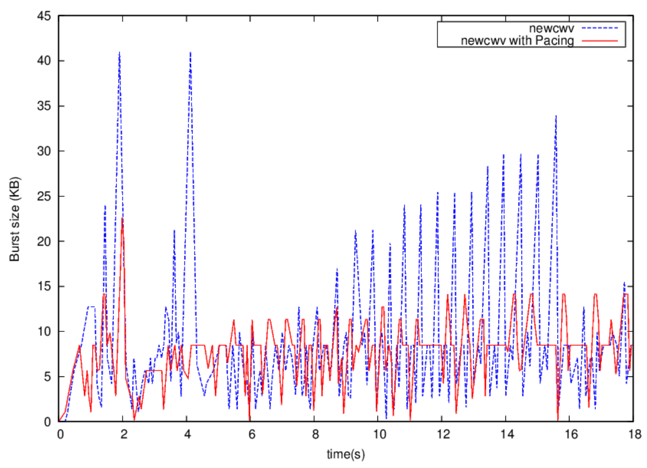

Figure 4. The effect of Pacing on a persistent TCP connection with a VLC client; newCWV without pacing (blue dashed line) has larger peaks than with pacing (red solid line) confirming the spread of packet bursts over time.

Figure 2 plots the TCP *cwnd* for a typical TCP connection using DASH-JS and VLC. The TCP sender grows the *cwnd* until it detects a congestion event (e.g. loss due to router buffer overflow). In general the *cwnd* for the VLC client remained higher compared to that for the DASH-JS client, due to the difference in segment request patterns (these are under the control of the DASH client).

Figure 3 compares the dynamics of *cwnd* in NewReno and newCWV with VLC for one of the four TCP connections that was used for 160s of the total video playtime (~600s). The subsequent connections also had the similar pattern. The *cwnd* in newCWV

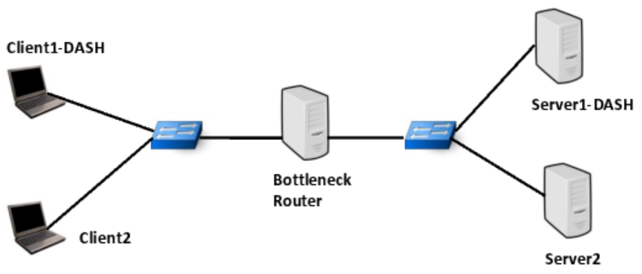

Figure 6. Experimental setup for multiple flows.

maintains a high value for a longer time compared to NewReno and reset only in the case of congestion. The NewReno sender collapses the *cwnd* almost twice as frequently as newCWV due to both congestion and *cwnd* reduction after a short idle period.

Because using newCWV, the *cwnd* is preserved during a rate-limited period, this allows the sender to transmit *cwnd* packets when the application changes the rate to transmit faster. If the *cwnd* is large, sending faster could lead to a burst of packets, and this could in turn result in increased loss. Pacing may therefore need to be introduced at the sender to address this burstiness.

Figure 4 shows the amount of data per second sent by the DASH server for newCWV and newCWV with pacing using the VLC player. Pacing results in a smoother flow with less variance from the average rate. This in turn results in fewer packet losses.

Figure 5 compares the number of packets lost in each one second period for NewReno, newCWV, and newCWV-pacing using VLC client. newCWV resulted in higher rates of loss because it allowed larger bursts to be sent. However, combining newCWV with pacing was shown to significantly reduce the rate of loss (Some residual packet loss is needed by TCP as congestion feedback when traffic exceeds the capacity being shared, and therefore loss cannot be entirely avoided).

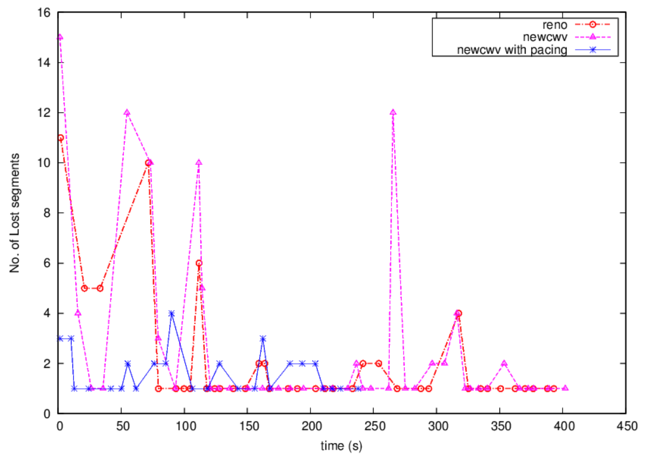

Figure 5. Packet loss at the bottleneck comparing NewReno, newCWV, and newCWV-pacing in a DASH server; newCWV with pacing experienced less loss than the other TCP variants.

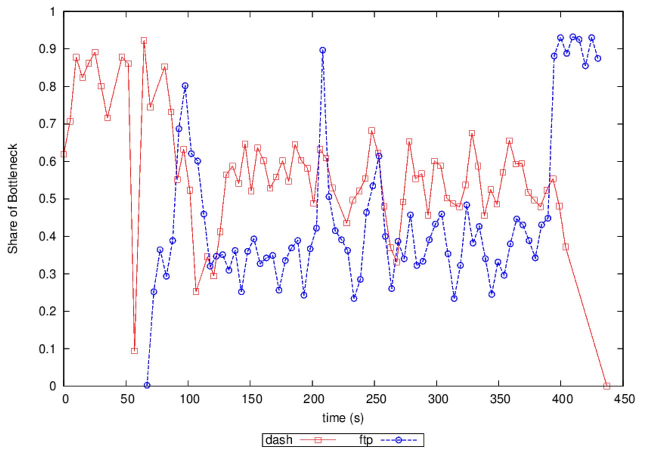

Figure 7. Share of the network bottleneck between NewReno FTP and newCWV-pacing-DASH.

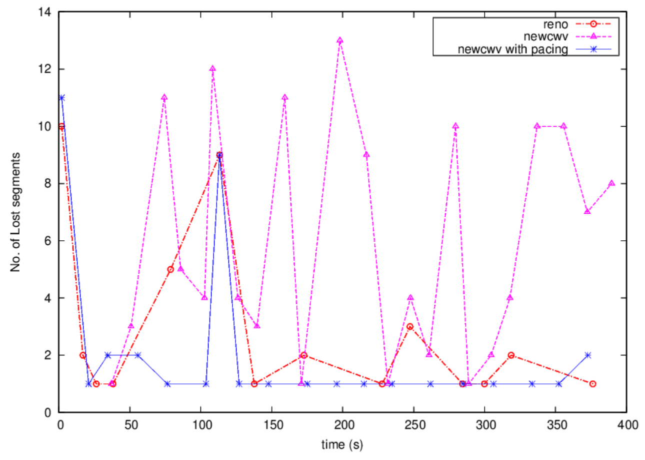

Figure 8. Packet loss comparing NewReno, newCWV, and newCWV-pacing.

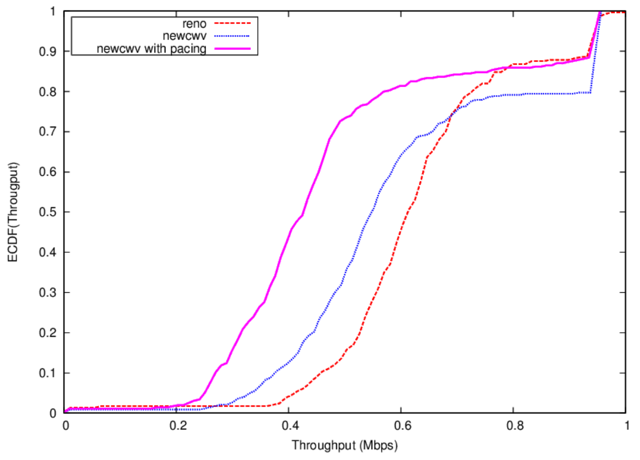

Figure 9. Throughput for NewReno FTP competing with the different DASH clients.

5. PATH SHARING WITH MULTIPLE FLOWS

This section presents a series of experiments to determine the effect of using newCWV with DASH when the DASH traffic co-exists with other types of traffic, and therefore needs to share the capacity of the network bottleneck.

The test bed shown in Figure 6 was used to introduce cross-traffic sharing the network bottleneck. The cross-traffic was generated by two clients and two servers. Client2 was used to generate FTP cross-traffic competing with the DASH flow from Server1 to Client1. The shared bottleneck was configured at 1Mbps. As expected, the DASH flow experienced more congestion in this multi-flow scenario than in the previous single flow experiment.

A VLC DASH client was used with persistence enabled, as described in Section 3.2. The TCP bulk cross traffic (FTP transfer) was started some time (1 min) after the start of DASH flow. The FTP traffic used TCP NewReno.

Figure 7 shows the fraction of bottleneck capacity used by DASH and FTP. At the start, only the DASH flow was present and was able to consume the whole capacity available along the network path. When the FTP flow started, the DASH client adapted its rate to share the available capacity with this other flow. The aggregate of the two flows almost equals the capacity of the bottleneck. When the DASH flow was stopped after 400s, the FTP flow increased its rate to around 90% of the bottleneck capacity.

Figure 8 presents the amount of losses over a one second period for the DASH flows with different algorithms. The occurrence of congestion was less and more controlled when newCWV was used and pacing was enabled. Fewer losses resulted in the DASH flow achieving a more stable behavior and adapting to a more predictable rate resulting in smoother playback of the video segments.

The DASH network traffic characteristics depend on the choice of the underlying transport protocol. With different congestion control algorithms, the server's response to congestion and the pattern of packets injected into the network are quite different. As a result, the cross-traffic sharing the bottleneck experiences different pathologies. The throughput achieved by the FTP application was affected by these choices.

Figure 9 presents the Empirical Cumulative Distributive Function (ECDF) curve showing the FTP rate over a 1 second interval for newCWV-pacing and NewReno. The curve shows a gap of around 0.25Mbps (250 Kbps), which means DASH with newCWV consumes more capacity than NewReno with DASH. The rest of the bottleneck capacity was available for use by the FTP application.

In the case of congestion following a non-validated period, newCWV tended to allow a higher *cwnd* than allowed by NewReno. This behavior ensures the cwnd is larger, benefiting bursty traffic, such as traffic from a DASH server. Therefore, in a congested path (1Mbps scenario), a sender using newCWV could potentially be more aggressive than Standard TCP. This behavior therefore needed to be explored to see if this effect is detrimental to other network traffic.

The observation was that the FTP performance was not significantly impacted when sharing with a DASH flow using newCWV, (importantly we did not observe any cases where newCWV caused the DASH flow to starve the competing flows preventing them from making progress when sharing with DASH traffic). Both types of flow were shown to co-exist and share the bottleneck for the entire duration. Similar results (not shown) were obtained for DASH with a competing HTTP flow.

Our results support incorporating newCWV as a standard feature, and that this sender-side change to the TCP stack can effectively replace the experimental approach advocated by TCP CWV. The experiments showed that fewer losses were observed when the packets entering the network are paced by the TCP sender at *cwnd*/RTT. Thus combining newCWV with pacing provides better network capacity sharing.

6. CONCLUSIONS AND FUTURE WORK

This paper has explored the use of two new TCP mechanisms in combination with widely used DASH clients. newCWV was evaluated with a persistent transport connection. TCP pacing was introduced to spread the transmission of TCP segments allowed by the *cwnd* across the duration of the estimated RTT, avoiding sending many segments contiguously at line-rate.

The experiments compared the behavior of NewReno, newCWV and newCWV with Pacing considering both single and multiple flow cases. The results show that newCWV with pacing results in an improved performance compared to TCP NewReno. Also, MPEG-DASH traffic was more stable and caused less loss. newCWV did not adversely affect other competing flows.

Importantly, the proposed changes to the TCP transport are only required at the HTTP server hosting the DASH content. The client application did not need to be aware of the changes to the TCP stack at the server to gain these benefits. Hence, this proposed change has a low deployment cost.

This paper summarises some interesting preliminary findings. In future, the proposed techniques will be evaluated with other commercial DASH clients through more rigorous experiments. We suggest that it would also be interesting to evaluate the fairness of the new techniques for DASH running with a set of other applications (like HTTP and other DASH flows etc).

7. ACKNOWLEDGEMENTS

This research was supported by the RCUK Digital Economy programme to the dot.rural Digital Economy Hub; award reference: EP/G066051/1; the EU FP7 OFELIA project (FP7-ICT-258365), the EU FP7 STEER project (FP7-ICT-318343) the NERC EVOp (NE-I002200-1) project and the RITE project (ICT-317700).

8. REFERENCES

[1] Cisco Systems. Cisco visual networking index: Forecast and methodology, 2012-2017. White Paper, May 2013.

[2] T. Stokhammer. 2011. Dynamic adaptive streaming over http stds and design principles. *ACM SIGMM Conference on Multimedia Systems (MMSys)*, February 2011.

[3] International Standard Organization (ISO). Information technology. Dynamic adaptive streaming over HTTP (DASH). *ISO-IEC 23009-1*, 2012.

[4] V. Paxson M. Allman and E. Blanton. 2009. TCP congestion control. *IETF RFC 5681 Standards Track*, 9 2009.

[5] A. Sathiaseelan, R. Secchi, G. Fairhurst, and I. Biswas. 2012. Enhancing TCP performance to support variable-rate traffic. *ACM CoNext Capacity Sharing Workshop*, Nice France, 2012.

[6] T. Huang et al. 2012. Confused, timid, and unstable: Picking a video streaming rate is hard, in *ACM conference on Internet Measurement*, pages 225-238, 2012.

[7] A. Mansy, B. V. Steeg, and M. Ammar. 2013. SABRE: A client based technique for mitigating the buffer bloat effect of adaptive video flows. *ACM Multimedia System Conference*, New York (USA), 2013.

[8] A. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. Begen.2013. Server-based traffic shaping for stabilizing oscillating adaptive streaming players. *ACM Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, New York (USA), 2013.

[9] G. Fairhurst, A. Sathiaseelan, and R. Secchi. Updating TCP to support rate-limited traffic. Technical report, *IETF Work-In-Progress, Internet draft*, 2013. draft-ietf-newcwv-03.txt.

[10] D. Wei, P. Cao, and S. Low. 2006. TCP Pacing revisited. *IEEE INFOCOM*, 2006.

[11] R. Fielding et al. Hypertext transfer protocol http/1.1. *IETF RFC 2068 Standard Track*, 1999.

[12] M. Handley, J. Padhye, and S. Floyd. 2000. TCP congestion window validation. *IETF RFC 2861*, 6 2000.

[13] A. Menon and W. Zwaenepoel. 2008. Optimizing tcp receive performance. *ATC'08 USENIX 2008 Annual Technical Conference*, pages 85-98, Boston (US), 6 2008.

[14] K. Kobayashi. 2006. Transmission timer approach for rate-based pacing TCP with hardware support. *PFLDnet*, 2 2006.

[15] A. Aggarwal, S. Savage, and T. Anderson. 2000. Understanding the performance of TCP pacing. *IEEE INFOCOM*, pages 1157-1165, 2000.

[16] D. Wischik. 2006. Buffer sizing theory for bursty TCP flows. In *International Zurich Seminar on Communications*, pages 98-101, 2006.

[17] Apache Software Foundation. Apache http server project. http://httpd.apache.org/

[18] Linux Foundation. Network emulator. http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

[19] Jonathan Corbet. 2013. TSO sizing and the FQ scheduler, August 28, 2013. http://lwn.net/Articles/564978/

[20] C. Timmerer S. Lederer, C. Muller. 2012. Dynamic adaptive streaming over http dataset. *ACM Multimedia System Conference (MMSys)*, pages 22-24, Chapel Hill (USA), 2 2012.

[21] ITEC. Dash-js a javascript and webm-based dash library for google chrome. http://www-itec.uni-klu.ac.at/dash/?p=792

[22] VideoLAN Organization. VLC media player. http://www.videolan.org/index.html

[23] Y. Chung, “Recent advancements in Linux TCP congestion control”, IETF 88 Vancouver, November 2013. http://www.ietf.org/proceedings/88/slides/slides-88-iccrg-6.pdf