Public Review for

Application Flow Control in YouTube Video Streams

Shane Alcock and Richard Nelson

Video traffic, either via wired or wireless networks, is growing fast and leading the demand for network bandwidth. For its major contribution on Internet traffic volume. Its control mechanism has a significant impact on the network performance. This work investigates a control mechanism employed by the most popular video service, YouTube, called "block sending" and shows that their mechanism si worse than vanilla TCP flow control. The paper pitcures a cautiounary tale of how a well-meant feature could actually have tne opposite effect. CCR readers should find it very interesting.

Public review written by
Sue Moon
KAIST, South Korea

acm sigcomm

Application Flow Control in YouTube Video Streams

Shane Alcock
University of Waikato
Hamilton, New Zealand
salcock@cs.waikato.ac.nz

Richard Nelson
University of Waikato
Hamilton, New Zealand
richardn@cs.waikato.ac.nz

ABSTRACT

This paper presents the results of an investigation into the application flow control technique utilised by YouTube. We reveal and describe the basic properties of YouTube application flow control, which we term *block sending*, and show that it is widely used by YouTube servers. We also examine how the block sending algorithm interacts with the flow control provided by TCP and reveal that the block sending approach was responsible for over 40% of packet loss events in YouTube flows in a residential DSL dataset and the retransmission of over 1% of all YouTube data sent after the application flow control began. We conclude by suggesting that changing YouTube block sending to be less bursty would improve the performance and reduce the bandwidth usage of YouTube video streams.

Categories and Subject Descriptors

C.2.0 [**Computer Communications Networks**]: General

General Terms

Measurement, Performance

Keywords

YouTube, Flow Control, Block Sending, Packet Loss, DSL

1. INTRODUCTION

YouTube [1] is a video-on-demand service that allows users to stream user-generated video content through their web browser. According to the Alexa traffic rank [2], YouTube is currently the third most popular website on the Internet and has been noted in literature as being one of the primary causes behind the recent increases in HTTP traffic observed in measurement studies [3]. As a significant contributor to traffic observed on the Internet, it is especially important that YouTube traffic patterns are understood and modelled correctly by Internet researchers. There has been a notable quantity of work examining client behaviour, e.g. trends in YouTube video popularity [4] [5], but there has been little research into the behaviour of the YouTube servers themselves. Rather, there seems to be an implicit assumption in the research community that YouTube traffic behaves in much the same way as any other large HTTP download.

This paper presents an initial look at the application flow control utilised by YouTube servers to conserve bandwidth and prevent the client connection from being saturated. We use passive packet header traces of YouTube traffic captured from both an academic and a residential DSL network for our analysis. The YouTube application flow control technique, which we term *block sending*, has not been previously described in literature. We derive and describe some properties of the block sending algorithm, including the block size and conditions for commencing block sending. We also examine how the algorithm performs in practice and identify instances where the YouTube flow control interacts poorly with the underlying TCP mechanisms, leading to increased congestion and packet loss.

2. BACKGROUND

For most TCP applications, the control provided by TCP is sufficient to ensure data is transmitted at a fair rate that will not overwhelm the receiver or any of the links along the path. Congestion control [6] prevents the sender from congesting the path to the receiver, while the receive window [7] ensures that the receiver itself is not overwhelmed with more data than it can handle. However, it may also be desirable for the application to provide additional flow control if it is not strictly necessary for the application data to reach the client as fast as TCP would otherwise allow. Instead, the application may limit the rate at which data is passed to the network stack for transmission. The TCP control mechanisms still apply, though, so the effect of the application flow control may be reduced if the connection is already limited by the receive or congestion window.

Video streaming applications, such as YouTube, are an obvious example where application flow control can be useful. Assuming the user watches the video from start to finish, there is little benefit in sending the client data far ahead of where they are currently viewing. Instead, the application can reduce its sending rate to ensure the client has enough data to play the video smoothly without congesting the network. As a result, other applications that the client is using can still achieve satisfactory throughput at the same time and the server can avoid sending unwanted data to the client if the user decides to cease watching the video early.

Application flow control can have a significant impact on the models of network traffic that are used in research, such as simulation studies. Conventional TCP traffic models will not produce a realistic traffic pattern for applications that do not conform to expected "greedy" sending behaviour and instead implement their own flow control. Therefore, it is important to measure and understand the application flow control techniques used by major TCP applications, such as YouTube, so that appropriate models can be developed.

Another reason for measuring application flow control is

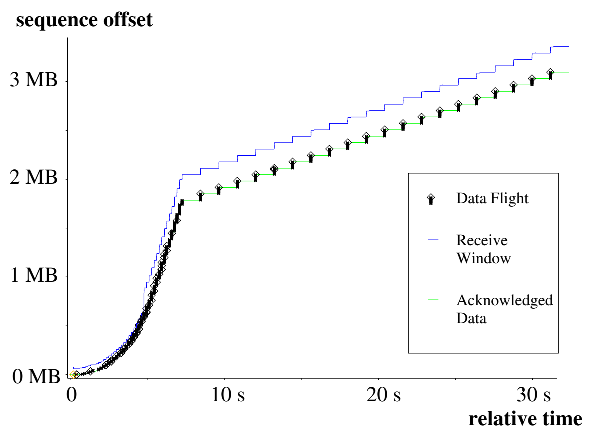


Figure 1: A time sequence graph of a YouTube video stream flow from the Auckland dataset.

to evaluate if it is performing as intended, particularly in parallel with the underlying TCP control mechanisms. Application behaviour that may seem like a good idea can have unintended consequences at the TCP level, leading to excessive packet loss, congestion window reductions and lessened throughput. Discovering and highlighting such problems will enable the application developers to rectify them, leading to improved application performance.

3. DATASETS

For this study, we have examined YouTube video streams from two different packet header trace sets from the WITS archive [8]. The Auckland data set was selected from the Auckland X trace set that was captured in October 2009 at the University of Auckland, New Zealand. At the time of capture, KAREN (the New Zealand research and education network) [9] peered directly with the Google autonomous system that YouTube is hosted within. As a result, all connections between Auckland and YouTube were over a research network with high bandwidth and capacity. This enabled us to examine YouTube behaviour in a context where congestion (and the resulting packet loss) was minimal.

The ISP data set was extracted from the ISP C-II trace set that was captured from a New Zealand ISP in January 2010. To examine the YouTube session behaviour where packet loss due to path congestion was common, we filtered the traces to only include residential DSL customers. In contrast to the Auckland data, connections between the ISP users and YouTube occurred over links with a much lower capacity and bandwidth-per-user and are more representative of the experience of the average New Zealand consumer.

We filtered the unencrypted packet traces to only include traffic to and from IP addresses that resolved to known hostnames for YouTube video servers. The IP addresses were provided by the authors of [10], meaning that local New Zealand caches have been excluded from our analysis [1].

The filtered traces were then processed using the flight analysis module included with *libtcpcsm* [11] to produce a record of observed TCP flights and loss events for each YouTube flow. Flows where the server transmitted less than

[1]Comparing local cache behaviour with the US-based servers is one avenue of future work.

Name	Duration	Start Date	Flows	Bytes
Auckland	7 days	2009/10/21	95,500	846 GB
ISP	7 days	2010/01/07	14,656	109 GB

Table 1: YouTube traffic observed in our datasets.

1 MB data were discarded, as these were too short to provide reliable information about the application flow control. For applications where the sender is unlikely to be waiting on user interaction, such as video streams, the flight sizes should match the largest amount of data that the sender can transmit within the limits of the congestion and flow control algorithms. We can therefore examine the sending behaviour of the YouTube servers by analysing the pattern of flights sent for each video stream. The resulting datasets are described in Table 1. The filtered YouTube datasets represent 7.4% and 3.9% of the TCP port 80 traffic observed in the Auckland and ISP traces respectively.

4. YOUTUBE FLOW CONTROL

Figure 1 is a time sequence graph created using *tcptrace* [12] that depicts a typical YouTube video stream flow from the Auckland dataset. The pattern of flights clearly changes approximately eight seconds into the flow. Prior to that point, the sending pattern conforms to the expected TCP slow-start behaviour, where the sending rate grows as data is successfully received and acknowledged by the client. Afterwards, the sender transmits data at a constant rate that is much slower that what had been achieved earlier. As no packets were lost, the sender should not be limited by its congestion window and the large gap between flights suggests that the TCP buffer has not been restricted. The receive window line also clearly indicates that the client can accept more data than the server is providing.

Therefore, we conclude that the YouTube application is responsible for the decreased sending rate. We observe that the application is writing consistently sized blocks to the network stack at a reduced rate to limit the amount of data that is sent to the client. The YouTube server sends the video as fast as possible for an initial buffering period before settling into the constant sending rate. This rate is probably at or slightly above the playback speed, so as to maintain the buffer and ensure smooth playback for the client while minimising the amount of bandwidth used by the flow.

Based on manual inspection of the flight records and time sequence graphs for individual YouTube flows, we summarise the primary characteristics of the YouTube application flow control (which we shall henceforth refer to as *block sending*) as follows:

- The gap between blocks greatly exceeds the inter-flight gaps observed during the initial buffering phase. This gap often exceeds one round-trip time (RTT).
- The packet rate when transmitting blocks is very high; flights sent during block sending have very short durations, i.e. time between the first and last packet.
- The block writes are typically a multiple of 64 kilobytes in size, with some rare exceptions (see §5.2).
- Flight sizes during block sending will never exceed the congestion or receive windows, so a block can be (and often is) split over multiple flights.

- The time between each block write is very consistent. Any additional delay in sending a block is compensated for when transmitting subsequent blocks, so that the constant sending rate is maintained overall.
- After packet loss, block sending usually concludes and the congestion window will dominate the sending pattern again. However, once the congestion window has recovered sufficiently, block sending may then resume.

5. ANALYSIS

To examine YouTube application flow control in more detail, we developed a tool to detect block writes from the flight records for each YouTube flow that we had extracted earlier using *libtcpcsm*. The block detection algorithm is based heavily on the block properties described in §4, albeit with many refinements to correct misclassifications identified during testing. In basic terms, the algorithm searches for the next flight that could be the start of a block based on the inter-flight gap and flight duration. A block may be made up of multiple flights, so the end of the possible block is determined by searching for either the next loss event or large inter-flight gap. Finally, the potential block is verified as to whether it is a genuine block, i.e. checking that flight sizes, inter-flight gaps, etc. within the block meet expectations. For example, one such criteria is that no individual flight within a block exceeds 64 KB and that the first flight is always the largest flight [2].

The minimum complete block size supported by our tool is 64 KB, as this was the smallest block size observed during our initial analysis. The tool can also detect instances where the server appeared to be sending a block but was interrupted by a loss event before 64 KB was sent. For each block detected, our tool reports the block size, duration, start time, number of prior loss events and the block transmission rate (BTR). The BTR is calculated by dividing the size of the block by the time difference between the start of the block and the start of the next one. If no block immediately follows the current block, the BTR is not reported. Our tool also calculates the median BTR for each flow. As we have defined block sending behaviour as having a consistent transmission rate, our tool requires at least half of the BTR values to be within 2% of the median to report a valid BTR for a flow.

5.1 Prevalence of Block Sending

Firstly, we investigated whether block sending behaviour is commonplace in both datasets, producing the results reported in Table 2. The rate of block sending in the ISP dataset was surprisingly low (only 56.7% compared with 87.6% in the Auckland data), so we investigated the flows that did not block send in greater detail to determine if there was a reason that could prevent the server from employing block sending. First, we used simple deep packet inspection to identify flows that were not FLV video streams and therefore unlikely to be using the YouTube application flow control. This was done by searching for a packet where the first three bytes of payload were "FLV". This was possible because the original trace files had retained four bytes of application payload for each packet.

[2]More detail about the algorithm can be found in the source code for the tool, which can be found at `http://www.wand.net.nz/~salcock/youtube/`.

	Auckland	ISP
YouTube Flows	95,500	14,656
Block Sending Flows	83732 (87.6%)	8313 (56.7%)
Not a Video Stream	1996 (2.1%)	602 (4.1%)
Recv. Win. Limited	309 (0.3%)	993 (6.8%)
Cong. Win. Limited	40 (0.04%)	2044 (13.9%)
High Loss Rate	231 (0.2%)	988 (6.7%)
Unclassified	9192 (9.6%)	1716 (11.7%)

Table 2: Number of flows in each dataset where block sending was detected. The additional categories describe possible situations where a flow is unlikely to block send.

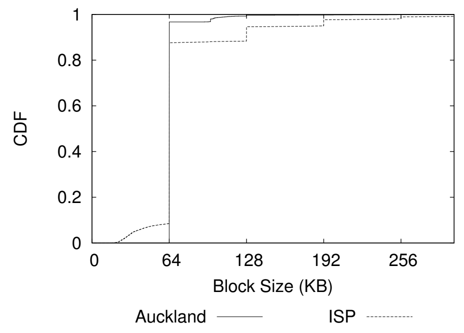


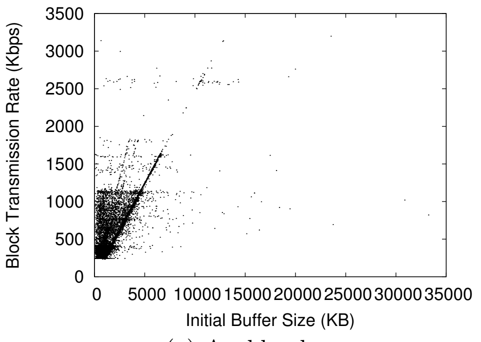
Figure 2: Distribution of the size of blocks sent by YouTube servers during the block sending phase. The X-axis has been truncated at 300 KB.

We also attempted to identify flows where the TCP control mechanisms were likely to prevent the sender from block sending in a detectable fashion. For instance, if the receive or congestion windows are too small, a flow will never achieve a sufficient transmission rate for block sending to be apparent. For the purposes of this study, flows where the receive window never exceeded 20 KB were classed as receive window limited. 20 KB was selected as a threshold to exclude all flows that would take more then 3 RTTs to transmit a typical 64 KB block, as most servers would fail to transmit the block before the next one appears in the queue.

Finally, flows that had a high rate of packet loss (we used an arbitrary value of less than 250 KB of new data per loss event as the threshold, based on observations of some obviously lossy flows in our data set) were classed as high loss flows. Table 2 describes the number of flows that matched each of these categories. Ignoring high loss, window limited and non-video flows, we see that the proportion of flows where our analysis tool detected block sending increased to 90.1% for Auckland and 82.8% for the ISP dataset. This suggests that block sending is standard practice for YouTube servers.

5.2 Block Sizes

Figure 2 shows the distribution of block sizes in the Auckland and ISP datasets. 64 KB is easily the most common block size, accounting for 97% of blocks in the Auckland data and 79% of blocks in the ISP data. Block sizes that are a multiple of 64 KB are also prominent in the ISP distribution, mainly due to delays in the path causing multiple 64 KB blocks to be merged and indistinct by the time they



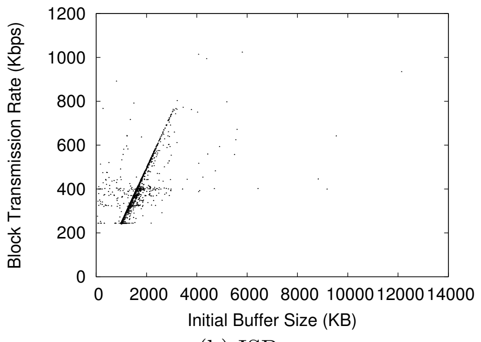


Figure 3: Scatter plots showing the relationship between the number of bytes sent during the initial buffering phase and the block transmission rate for each YouTube flow.

reached the passive monitor. Blocks smaller than 64 KB, as observed in the ISP data, were blocks that were interrupted by packet loss before the block was completed. Nearly 10% of all blocks in the ISP data were interrupted in this fashion. Finally, we note that a small proportion of blocks in the Auckland data range between 96 and 128 KB in size. Manual validation has shown that the blocks were correct, but we are yet to find a suitable explanation.

We believe that the 64 KB block size arises from the video files being stored on servers using the Google File System (GFS) [13]. GFS divides files into 64 MB chunks, which are then further split into 64 KB blocks for checksumming purposes. To maximise performance, a YouTube server reads and writes the 64 KB blocks directly wherever possible, producing the results shown in Figure 2.

5.3 Initial Buffering

We examined the initial buffering period to determine the conditions that must be met for a YouTube server to commence block sending. We deemed the initial buffering period to be over once our analysis tool detected a block (complete or interrupted). There was no consistent flow duration or bytes sent before each flow began block sending, but we did find that there was a strong linear relationship between the number of bytes sent during the buffering phase and the median block transmission rate, as shown in Figure 3.

Only flows for which we could determine a valid median BTR are represented in Figure 3. This proved problematic for the ISP dataset, as many blocks were immediately followed by packet loss events that ended block sending (we examine this problem in §5.4), preventing us from calculating the BTR. As a result, only 45% of the 8313 ISP flows are represented in Figure 3(b). By contrast, 81% of block-sending flows from the Auckland data are shown in Figure 3(a). We see that video streams utilising a higher bitrate during block sending transmit a proportionately larger amount of data during the buffering phase. The Auckland data has some horizontal banding, which may be due to a minimum block transmission rate being used for some flows.

For each flow, we also divided the initial buffer size by the median BTR to calculate the time that would have been required to transmit the initial buffer using block sending. The distribution of those values is presented in Figure 4, which shows that the initial buffer was equivalent to 32 seconds of block sending for the vast majority of YouTube flows

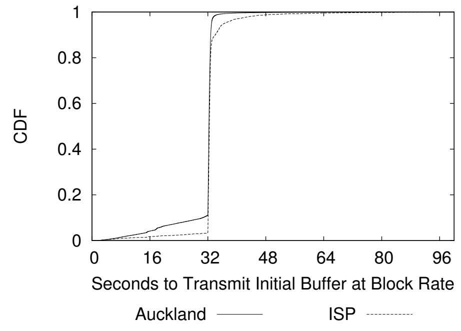


Figure 4: CDF of the time needed to transmit the initial buffer for a YouTube flow using the subsequent block sending rate. The X-axis has been truncated at 100 seconds.

in both datasets. This proves that the relationship between the BTR and the initial buffer size depicted in Figure 3 is the same for both Auckland and the ISP, although we have not been able to determine the exact reasoning behind the selection of 32 seconds as a threshold.

5.4 Interaction with TCP

Figure 5 is a time sequence graph showing part of a flow from the ISP dataset. The first 12 seconds of the flow (which we have mostly omitted) covers the initial buffering period. The first block can be identified by the long pause prior to a large burst of packets. This is immediately followed by a packet loss event, as indicated in the graph, which results in the TCP congestion control reverting to slow start. In this context, a packet loss event refers to a retransmit that leads to TCP loss recovery (either a fast retransmit or a retransmit timeout). Multiple packets may be retransmitted in response to a loss event. Once the congestion window has grown again, there are two more attempts at block sending which are both immediately followed by packet loss as well.

Therefore, we suggest that block sending is responsible for the packet loss, especially given that no loss was observed during the initial buffering phase. The application delays pushing the block onto the TCP stack, meaning that the congestion and receive windows are empty by the time the block is written. As a result, the entire block is transmit-

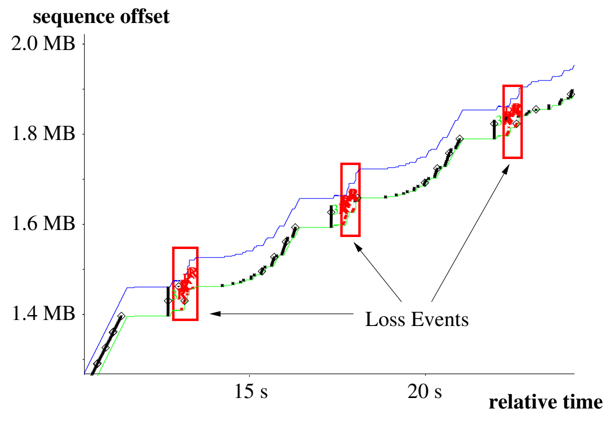


Figure 5: A time sequence graph showing a YouTube flow from the ISP dataset. Every attempt at block sending is immediately followed by packet loss.

ted immediately as a burst of packets, effectively creating congestion. The idle time prior to writing the block is not long enough for congestion window reduction [14] to be employed, yet other competing flows may have increased their congestion window in the interim, further contributing to path congestion.

This short-term congestion leads to a much higher probability of packet loss, resulting in otherwise avoidable retransmissions as well as forcing the TCP implementation to reduce the congestion window, lessening the throughput for the video stream until the window recovers (note that this does not necessarily mean that the playback buffer has recovered!). If this problem is widespread, it would suggest that the YouTube application flow control may be detrimental, rather than beneficial, to the client experience and should be re-evaluated by the YouTube developers.

We have used two different definitions to determine if a packet loss event in our datasets was caused by block sending. The first and strictest definition states that a loss event was caused by block sending if preceded by one (and only one) block, similar to the events depicted in Figure 5. Henceforth, we shall refer to this definition as definition A. The second definition (definiton B) states that if a loss event is preceded by either one or two blocks then it was caused by block sending. The reasoning is that the first block can often be transmitted without packet loss but will still create congestion that leads to the following block being disrupted.

Table 3 presents the proportion of loss events and retransmissions that can be attributed to the YouTube application flow control, along with some general statistics about the overall loss and block rates. For this analysis, we have ignored loss events, retransmits and bytes sent prior to the first block observed for each flow, i.e. the initial buffering period. Also, when calculating the number of bytes retransmitted, we have assumed a worst-case scenario of 1500 bytes per retransmitted packet, as this is the MTU commonly employed by YouTube clients.

The results show that, unsurprisingly, loss events and retransmissions are much more common in the ISP data. ISP flows also transmitted fewer blocks than Auckland flows, presumably because the higher loss rate frequently disrupted the block sending process. Up to 40% of all loss events and nearly half of all retransmits in the ISP dataset could

	Auckland	ISP
Block Sending Flows	83,732	8,313
Avg. Megabytes per Flow	8.13	6.53
Avg. Loss Events per Flow	2.13	21.91
Avg. Blocks per Flow	115.88	18.10
Avg. Blocks per Loss Event	54.3	0.83
Avg. Retxs per Flow	9.94	139.15
Avg. Retxs per Loss Event	4.66	6.35
Loss Events matching def. A	5.8%	27.8%
Loss Events matching def. B	9.8%	40.1%
Retxs matching def. A	4.6%	36.0%
Retxs matching def. B	8.1%	47.5%
Bytes retransmitted (def. A)	< 0.01%	1.1%
Bytes retransmitted (def. B)	< 0.01%	1.5%

Table 3: Statistics describing the proportion of loss events and retransmits that can be attributed to block sending using definitions A and B.

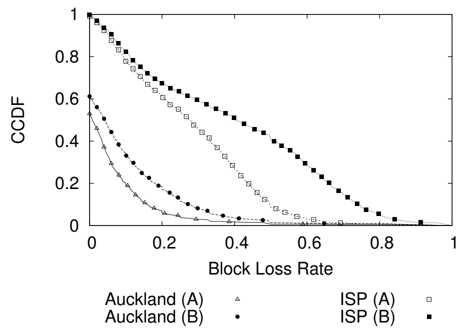


Figure 6: CCDF of loss events against the flow block loss rate.

be attributed to congestion caused by block sending. This resulted in between 1 and 1.5% of the YouTube traffic in the ISP dataset being retransmitted due to block sending. The impact on the Auckland data was not negligible either; nearly 10% of loss events were attributable to block sending, although the overall proportion of traffic retransmitted was very low. However, this suggests that the bursty nature of block sending can still create congestion even in excellent network conditions.

Finally, we examined the block loss rate (BLR), which we defined as the proportion of loss events for a given flow that could be attributed to block sending. Figure 6 is a CCDF showing the distribution of loss events in terms of the BLR of the flow that they occurred in. The Auckland results show that the vast majority of loss events occurred in flows with a low BLR value, meaning that block sending was seldom the leading the cause of packet loss in Auckland flows. A significant proportion of loss events occurred in flows where the BLR was zero, i.e. no loss events could be attributed to block sending.

By contrast, almost all ISP flows have a BLR greater than zero; block sending was responsible for at least one loss in almost every flow where block sending was detected. Also, the impact of block sending is much greater in the ISP dataset compared with Auckland. Using definition B, over 40% of loss events occurred in flows where the BLR is greater than 0.5. This suggests that many residential DSL users would see a significant improvement in YouTube performance if the application flow control was changed to be less bursty.

6. RELATED WORK

Unsurprisingly, YouTube has been a popular research topic in the Internet measurement community recently. Much of this work has focused on the characteristics of YouTube content, such as file size, bit-rate and popularity. Examples of such work are [4], [5], [15] and [16]. By contrast, our research has examined individual YouTube flows in greater detail to investigate how the YouTube content is delivered to clients and characterise the behaviour of the YouTube server application. There is some overlap, though; for instance, our results suggested there is a much broader range of video bit-rates than demonstrated previously. We suspect this is due to YouTube supporting a greater variety of video resolutions than when the earlier studies were conducted.

[17] noted that YouTube content delivery is rate limited, with a maximum transfer rate of approximately 1.25 Mbits/s. Our work elaborates on this finding by revealing how the rate limiting is performed and evaluating the effect on streaming performance. The authors of [17] also found that many YouTube transfers achieved low throughput over DSL connections but claimed that the application was not responsible for the poor performance. Our results disagree with the latter claim, as we have found that block sending by the application can cause congestion and packet loss, particularly for DSL clients, which may account for the low throughput observed by [17]. Finally, [10] developed a technique for identifying traffic flows for various video streaming applications, including YouTube. The authors generously provided us with their list of YouTube server IP addresses which we used to create our own datasets.

7. CONCLUSION

There are two principal conclusions that we draw from this research. Firstly, YouTube implements a previously undocumented form of flow control at the application level, which we call block sending, that operates in addition to traditional TCP flow control mechanisms. We have been able to detect block sending in over 80% of YouTube flows that we examined, from both residential DSL and academic networks. We have also presented research that examines block sending in more detail, showing that the blocks are typically 64 KB in size (matching the block size used by GFS) and that the amount of data sent during the initial buffering period is equivalent to 32 seconds of block sending.

The second conclusion is that block sending can have a detrimental effect on YouTube flow performance, particularly if the client is streaming the video over a congested link. Blocks are typically transmitted as a large burst of packets, creating additional congestion and often leading to packet loss and significantly reduced throughput. Our analysis showed that over 40% of the packet loss events observed by YouTube clients using residential DSL could be attributed to congestion caused by block sending. These loss events resulted in a data retransmission rate of 1.5% of all bytes sent once block sending began. Given the popularity of YouTube, this is a significant quantity of data.

We have presented these results to engineers at YouTube and their parent company, Google. They have acknowledged that this is a legitimate problem and are currently working on modifying the block sending algorithm to be less bursty. We believe that this will offer improved YouTube performance for users and reduce YouTube's bandwidth requirements. The largest improvements will be seen by YouTube clients using congested connections, but well-connected clients should also see some benefit.

8. REFERENCES

[1] "YouTube," http://www.youtube.com.

[2] Alexa, "Top 500 Global Sites," http://www.alexa.com/topsites.

[3] G. Maier, A. Feldmann, V. Paxson, and M. Allman, "On Dominant Characteristics of Residential Broadband Internet Traffic," in *IMC '09: Proc. of the 9th ACM SIGCOMM Conference on Internet Measurement Conference.* New York, NY, USA: ACM, 2009, pp. 90–102.

[4] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System," in *IMC '07: Proc. of the 7th ACM SIGCOMM conference on Internet measurement.* New York, NY, USA: ACM, 2007, pp. 1–14.

[5] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Characteristics of YouTube Network Traffic at a Campus Network - Measurements, Models, and Implications," *Comput. Netw.*, vol. 53, no. 4, pp. 501–514, 2009.

[6] M. Allman, V. Paxson, and E. Blanton, "RFC 5681 - TCP Congestion Control," September 2009.

[7] J. Postel, "RFC 793 - Transmission Control Protocol," September 1981.

[8] WAND Network Research Group, "WITS," http://www.wand.net.nz/wits/.

[9] "KAREN: Kiwi Advanced Research and Education Network," http://karen.net.nz/home/.

[10] T. Mori, R. Kawahara, H. Hasegawa, and S. Shimogawa, "Characterizing traffic flows originating from large-scale video sharing services," in *Traffic Monitoring and Analysis: Second International Workshop, TMA 2010.* Springer, 2010, pp. 17–31.

[11] WAND Network Research Group, "libtcpcsm," http://research.wand.net.nz/software/tcpcsm.php.

[12] S. Ostermann, "tcptrace," http://www.tcptrace.org/.

[13] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 29–43, 2003.

[14] M. Handley, J. Padhye, and S. Floyd, "RFC 2861 - TCP Congestion Window Validation," June 2000.

[15] X. Cheng, C. Dale, and J. Liu, "Statistics and Social Network of YouTube Videos," in *Proc. of IEEE IWQoS*, 2008.

[16] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "YouTube Traffic Characterization: a View from the Edge," in *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement.* New York, NY, USA: ACM, 2007, pp. 15–28.

[17] L. Plissonneau, T. En-Najjary, and G. Urvoy-Keller, "Revisiting Web Traffic from a DSL Provider Perspective: the Case of YouTube," in *Proc. of the 19th ITC Specialist Seminar*, 2008.