Characterizing Netflix Bandwidth Consumption

Jim Martin*, Yunhui Fu*, Nicholas Wourms*
*School of Computing,
Clemson University,
Clemson, SC 29634
{jmarty,yfu,nwourms}@clemson.edu

Terry Shaw†
† Cable Television Laboratories, Inc.,
858 Coal Creek Circle
Louisville, CO 80027
t.shaw@cablelabs.com

***Abstract*—The widespread deployment and adoption of the Dynamic Adaptive Streaming over HTTP (DASH) standard is making Internet video-on-demand a 'standard' Internet application similar in impact as email and web browsing. While video streaming has been widely deployed and studied for decades, DASH-based streaming is very different as it involves adaptation both by the application and by TCP. The dynamics and implications of multiple levels of end-to-end congestion control are not well understood. The contribution of the research presented in this paper is twofold: first we characterize the bandwidth consumption of a widely deployed DASH application (i.e., Netflix); second we provide insight in how different implementations and different access networks can impact bandwidth consumption. Our results suggest that Netflix adaptation appears to default to TCP control during periods of heavy, sustained network congestion. However, the application algorithm is clearly intertwined with TCP control during periods of volatile network conditions. In one network scenario we observed a backlogged TCP flow achieve a throughput of 20.5 Mbps while a Netflix session (under similar path conditions) consumed less than 3 Mbps of bandwidth.**

***Index Terms*—DASH, Netflix, Video Streaming, Application Congestion Control**

I. Introduction

Video streaming has been widely deployed and studied for decades. In the early years, Internet streaming typically involved UDP and RTP/RTSP protocols. The application was likely a real-time video session which, due to the sensitivity to packet latency and jitter, never became widely popular. Internet-oriented Video-on-Demand (VoD) began as simple file downloads. As the popularity of watching video over the Internet has increased, VoD methods have evolved. Progressive downloading, which allows video playback to begin before all the data arrives, has been replaced recently by dynamic adaptive VoD techniques that allow the choice of video content to be changed on-the-fly. Adobe, Apple, Microsoft, Netflix, and the 3GPP wireless communities have all developed proprietary methods for adaptive TCP-based streaming. In 2009, the ISO solicited proposals for a standards-based framework to support Dynamic Adaptive Streaming over HTTP (DASH). The standard was ratified in December of 2011 [1], [2]. The 3GPP community recently finalized its DASH variant that is optimized for wireless environments [3].

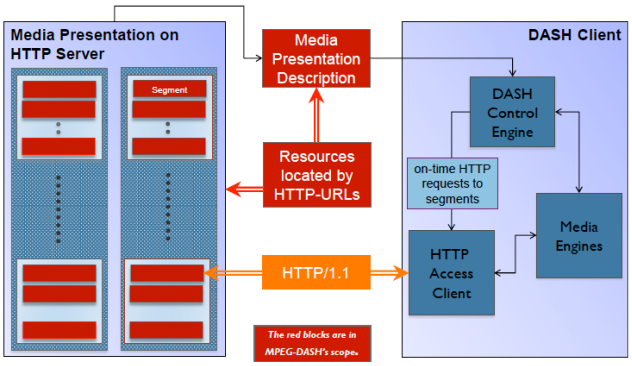
Ongoing academic research is providing foundations for understanding how DASH applications behave and how they might be improved. As a part of our ongoing research in resource allocation issues in broadband access networks, we seek to understand the impact of adaptive applications on congestion and bandwidth control mechanisms throughout the Internet or within a broadband access network. The work presented in this paper provides foundations for achieving this goal. The contribution of the research presented in this paper is twofold: first we characterize the bandwidth consumption of a widely deployed DASH application (i.e., Netflix); second we provide insight in how different implementations and different access networks can impact bandwidth consumption.

This paper is organized as follows. The next section provides an overview of the DASH protocol. The following section summarizes the methodology used in our study. We then present the results of the measurement analysis. The final section highlights our conclusions and identifies future work.

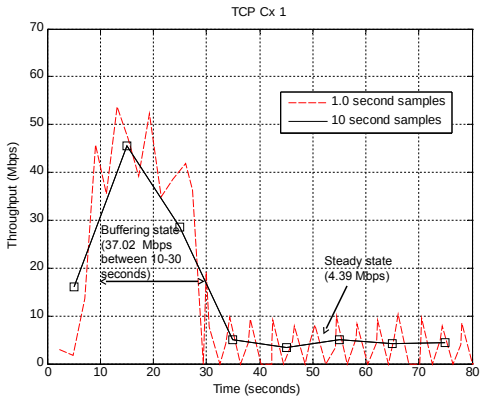
II. DASH Protocol

Figure 1a illustrates the main components of a DASH-based video distribution system. Multimedia content is encoded to one or more representations (different bit rates), allowing the client (i.e., the media player) to request any part of a representation in units of variable size blocks of data. The data content is made available to client devices by standard HTTP requests. The DASH server-side can be as simple as a standard HTTP web server. The available representations are maintained in a summary file referred to as the Media Presentation Description (MPD). MPD describes media types, resolutions, a manifest of the alternate stream and its URLs, minimum and maximum bandwidths, and required digital rights management (DRM) information.

As illustrated in Figure 1a, the DASH client contains three main functional areas: the HTTP access, the media engine that decodes and renders the video stream, and the control engine. The latter component monitors the arriving video stream and determines when a lower or higher quality stream should be requested. The client will maintain a playback buffer that serves to smooth variable content arrival rates to support the video playback. The client requests a new segment (tagged with the desired bitrate) once the playback buffer drops below a certain threshold. When the playback buffer is almost empty, the client is likely to go into a 'buffering' state where it requests segments at a high rate. The session therefore operates in one of two states: buffering or steady state. While in a



(a) DASH system components



(b) DASH client behavior

Fig. 1. Dash system components and client behavior

buffering state, the client requests data at a rate up to the available bandwidth over the path between the server and the client. If conditions permit, the session will attempt to find a 'steady state' where it requests segments at a rate necessary to playback the content at a given encoded bitrate. In the analysis we present in this paper, we define steady state somewhat arbitrarily as a time interval of at least 50 seconds where the session exhibits a stable level of bandwidth consumption. This assessment of steady state bandwidth consumption is a network-oriented measure and is quite different from an application level assessment. We leave for future work the more challenging issue of assessing the perceived quality of the rendered video.

Figure 1b illustrates the observed behavior of a Netflix client operating on a Windows desktop located at our campus. The results illustrate the evolution of the first 80 seconds of the single TCP connection involved with the Netflix session over an uncongested high speed network (i.e., the available bandwidth significantly exceeds the requirements of a Netflix session). The results capture the downstream throughput (i.e., the rate at which Netflix video data is forwarded to the client device). The solid curve formed by the square data points is based on 10 second throughput sampling intervals and the lighter dashed curve is based on 1 second samples. The figure shows the initial buffering state and, once the playback buffer is full, that the session moves into a steady state within about 35 seconds. While this basic 'waveform' associated with DASH is similar across different client devices that we have studied, we will show in our results that the detailed behavior of each client device varies.

III. Methodology

Figure 2 illustrates the measurement testbed that was used to monitor live Netflix sessions. We created a private network that connected four different client devices: an Xbox, a Roku, a Windows desktop, and an Android smartphone. The devices were connected to the campus network either through a private gigabit Ethernet or through an 802.11n network that was in turn directly connected to the campus network through our private gigabit Ethernet. A high-definition monitor (capable of 1080p format) was connected to the Xbox and Roku's HDMI port. We monitored Netflix traffic by obtaining network traces using tcpdump at the Linux Router. The router connects the measurement testbed with our campus Network which provides a 10 Gbps connection to the commodity (public) Internet. We utilized the Linux *netem* feature to apply a desired level of uncorrelated packet loss on inbound packets (i.e., packets from the Netflix server to the client) associated with the Netflix session under observation. We installed several additional Linux boxes in the testbed to serve as background traffic generators or to provide additional performance metric data. Linux Traffic Generators 1 and 2 were desktop PCs running Linux. Traffic Generator 3 was a laptop running Linux.

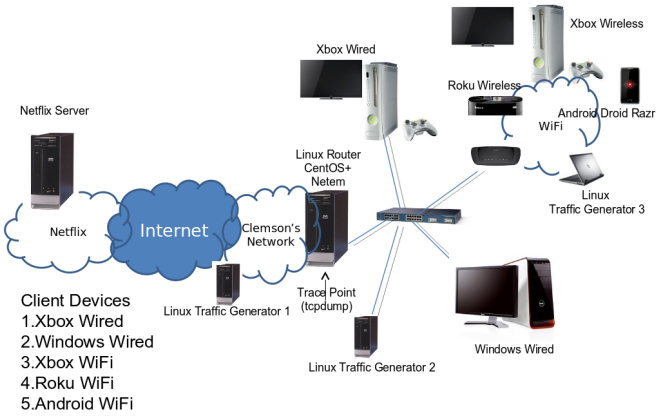


Fig. 2. Measurement testbed

We collected traces for a period of six months from January 2012 through June 2012. Our client devices interacted with Netflix servers located on different networks. For example, an Android WiFi device exchanged a small amount of data with the network www.netflix.com (operated by Amazon's EC2 service). However, the majority of traffic was from the Content Distribution Networks (CDN) including LimeLight, Level 3, and Akamai. As confirmed in [4], Netflix engages multiple CDNs; further, Netflix clients may decide to change CDNs in real-time if conditions are poor.

To ensure that the external network (our campus network, the Internet) and the Netflix content distribution system were not contributing unwanted network congestion we performed several 24 hour ping assessments with a Netflix server located in a CDN. One result over a 24 hour period (consistent with other tests run throughout our data collection period) indicated a loss rate of 0% and an RTT of 14.55 milliseconds (ms) with a standard deviation of 0.64 ms.

TABLE I
MEASUREMENT SCENARIOS

ID	Scenario Description
1	Ideal with 3% artificial loss (5 minutes no impairment, 5 minutes 3% loss, 5 minutes no impairment).
2	Ideal with 40% artificial loss (5 minutes no impairment, 5 minutes 40% loss, 5 minutes no impairment).
3	Stepped loss (200 seconds at 0% loss, 100 seconds at 5% loss, 100 seconds at 10% loss, 100 seconds at 0% loss).
4	Chaotic loss (300 seconds at 0% loss, 300 seconds at variable loss, 300 seconds at 0% loss).

Table I identifies four experimental scenarios that we have considered. For the results reported in this paper, each scenario consists of a set of experiments. For a given scenario the experiments varied the choice of client device or access network. In each experiment, we collected a trace of all packets associated with the Netflix session that was under observation.

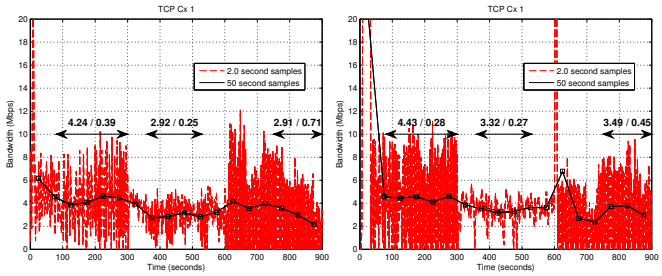
The results visualized in Figure 1b involve a scenario where only one TCP connection was active. It is more likely however for a Netflix session to utilize multiple (sometimes many) TCP connections. Some might be short-lived and others might be long-lived. Therefore, the results shown in the following section visualize the aggregate bandwidth consumed by all TCP flows associated with the Netflix session under observation. To do this, we developed an analysis tool that computes the total number of bytes sent to the Netflix client per interval (we used 2 second intervals). The byte count includes TCP and IP header overhead.

Scenario 1 depicts a network that experiences a temporary increase in packet loss. Scenario 2 depicts a temporary network outage event. Scenarios 3 and 4 provide insight in how the algorithm adapts to more complex periods of network congestion episodes that involve stochastic processes that replicate non-stationary network behavior.

IV. MEASUREMENT RESULTS

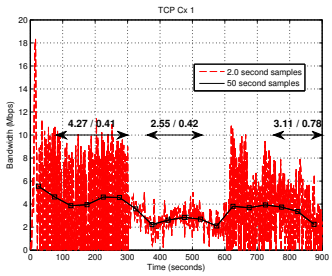
A. Scenario 1 results

Figure 3 illustrates the results from the five experiments involved with Measurement Scenario 1. As in Figure 1b, two curves are plotted in each sub-figure. The curve formed by the light dashed line is based on bandwidth samples obtained using 2 second intervals. The curve formed by the solid line (with the square points) is based on samples obtained using 50 second sample intervals. The figures identify the mean and standard deviation of bandwidth consumption during time intervals that we interpret to be periods of steady state.

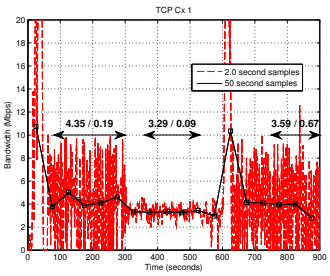


(a) Trace 1-1 (Xbox Wired)

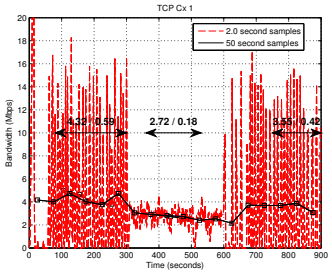
(b) Trace 1-2 (Windows Wired)



(c) Trace 1-3 (Xbox WiFi)



(d) Trace 1-4 (Roku WiFi)



(e) Trace 1-5 (Android WiFi)

Fig. 3. Measurement Scenario 1 results

Figures 3a and 3c plot the bandwidth consumed by the Xbox device when operated over a wired and wireless network respectively. The Xbox clients behave similarly in both scenarios. Figure 3a illustrates that the Xbox Wired device consumed 4.24 Mbps, 2.92 Mbps, and 2.91 Mbps in each of the three steady states. The Xbox WiFi device consumed slightly more bandwidth (4.27 Mbps, 2.55 Mbps, and 3.11 Mbps respectively). In both cases, the Xbox clients used two TCP connections in parallel. The aggregate bandwidth consumed is evenly split between the two connections.

The Windows Wired results, illustrated in Figure 3b, exhibits the largest bandwidth consumption during the initial playback buffer fill at time 0 through 50 seconds. The client consumed 40 Mbps of bandwidth for approximately 30 seconds. It converged to an initial steady state of 4.43 Mbps.

Figure 3d illustrates that the Roku WiFi device converged to steady state throughputs of 4.35 Mbps, 3.29 Mbps, and 3.59 Mbps. Further analysis (not presented in this paper) indicates that in the time period 550 to 600 seconds, a sequence of 5 short-lived TCP connections were involved. Each persisted for roughly 20 seconds. Once the loss process ended, a single connection was established that transferred all subsequent content.

Finally, Figure 3e visualizes the behavior of the Netflix client on the Android WiFi device. Once loss started at time 300 seconds, multiple TCP connections were observed. As with the Roku WiFi device, the connections were started sequentially and lasted about 20 seconds. The aggregate bandwidth consumed during this time period was 2.72 Mbps.

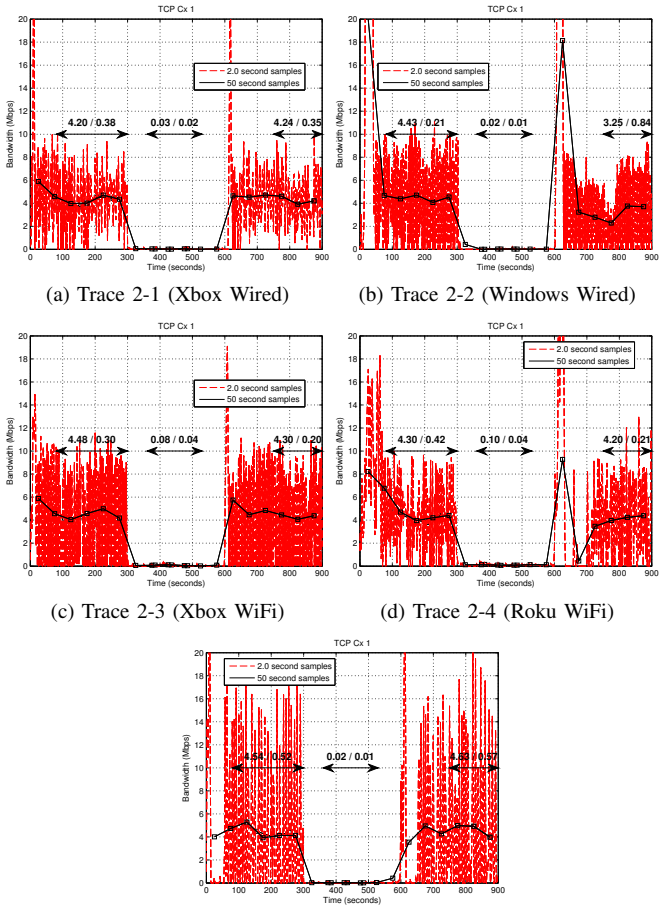
Based on the results from Measurement Scenario 1, the range of steady state throughputs achieved when loss was active varied from 2.55 to 3.32 Mbps. We determined that this is less than the TCP expected throughput of a backlogged connection (i.e., an always on application) when subject to similar conditions. We set *netem* to apply a 3% loss process and to add an artificial delay so the RTT between our Windows Wired device and the Linux Traffic Generator 1 shown in Figure 2 was around 16 ms which is approximately the measured RTT between our client devices and the Netflix data servers. In an additional experiment, we ran *iperf* between the traffic generator and the Windows Wired device ensuring that no other application traffic was being transferred by the Linux Router. Based on multiple 300 second transfers, the average TCP throughput was 6.11 Mbps (standard deviation 0.34 Mbps). This suggests that Netflix adaptation is the dominant control mechanism during sustained periods of moderate levels of network congestion.

Additionally, we make the following observations:

- The Roku, Windows desktop, and Android smartphone client devices use a single TCP connection at a time. Only the Xbox clients used two TCP connections concurrently, but the aggregate TCP throughput was always less than 4.3 Mbps.
- The range of initial steady state throughput (i.e., after the initial buffer fill) was consistent over all clients ranging from 4.24 Mbps to 4.43 Mbps.
- After the loss process was terminated, we observe that all devices converged to a steady state lower than the initial steady state. We repeated experiment 3 of Scenario 1, but ran it for 1800 seconds rather than 900 seconds. The throughput never exceeded 3.9 Mbps in the time interval between 600 and 1800 seconds.
- We have done additional tests to access the variability of the steady state statistics. Over short time scales (minutes), we found the 99% confidence interval range is within 5% of the mean.

B. Scenario 2 results

Figure 4 illustrates the behavior of the devices when subject to a temporary network outage. All devices were unable to receive content while the loss process was active (during the time interval 300 to 600 seconds). We measured the time from when the network outage began until when we observed that the video rendering had stopped. This time, which represents the size of the client device's playback buffer in units of seconds, ranged from 30 seconds (on the Android WiFi device) to 4 minutes (on the Windows Wired device). Once the outage ended (at time 600), the devices move into the buffering state to refill the playback buffer. As examples, we observed that the Windows Wired device experienced a peak bandwidth consumption of 40 Mbps and the Xbox devices experienced a peak of 15 Mbps. Each device used a new TCP connection once the outage was complete. The Xbox Wired device converges quickly to a steady state of 4.24 Mbps (the Xbox WiFi device converged to 4.30 Mbps). This recovery is quite different from that observed in the previous scenario where the Xbox Wired device converged to a steady state of 3.49 Mbps. The Windows Wired device enters a steady state of 3.25 Mbps from time 750 to 800 seconds and then switches to a steady state of 3.8 Mbps after time 800 seconds. The Roku WiFi device (Figure 4d) converges to a steady state of 4.20 Mbps, however it suffered an impairment at time 675 seconds when the consumed bandwidth dropped to less than 1 Mbps. The Android WiFi device converged to the highest observed steady state bandwidth consumption level of 4.63 Mbps. We conjecture that the device refills its playback buffer during this time period.



(a) Trace 2-1 (Xbox Wired)

(b) Trace 2-2 (Windows Wired)

(c) Trace 2-3 (Xbox WiFi)

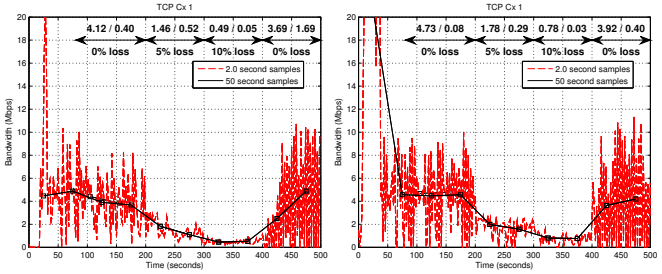
(d) Trace 2-4 (Roku WiFi)

(e) Trace 2-5 (Android WiFi)

Fig. 4. Measurement Scenario 2 results

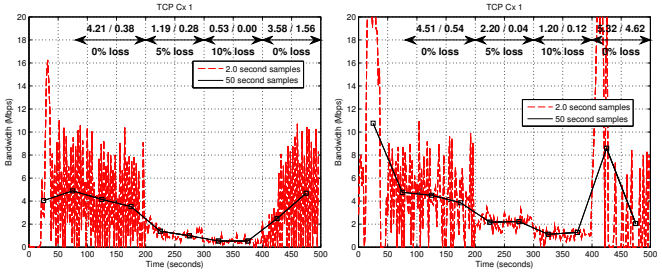
C. Scenario 3 results

Figure 5 illustrates the results associated with Measurement Scenario 3 where the artificial loss rate slowly evolves over



(a) Trace 3-1 (Xbox Wired)

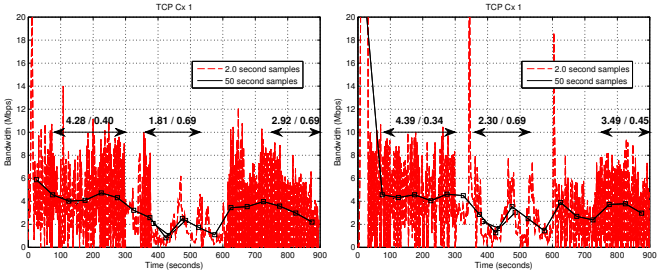
(b) Trace 3-2 (Windows Wired)



(c) Trace 3-3 (Xbox WiFi)

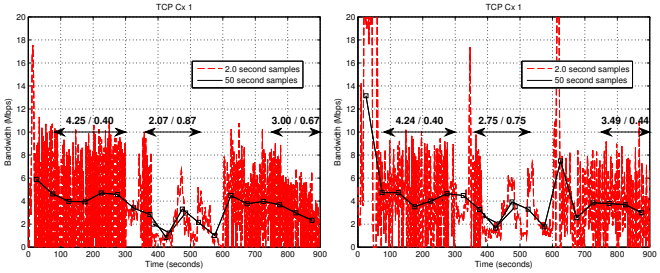
(d) Trace 3-4 (Roku WiFi)

Fig. 5. Measurement Scenario 3 results



(a) Trace 4-1 (Xbox Wired)

(b) Trace 4-2 (Windows Wired)



(c) Trace 4-3 (Xbox WiFi)

(d) Trace 4-4 (Roku WiFi)

Fig. 6. Measurement Scenario 4 results. The loss rate transitions during 0-900 seconds are: 0-300: 0%, 300-320: 4%, 320-340: 3%, 340-360: 0%, 360-380: 0%, 380-400: 8%, 400-420: 10%, 420-440: 9%, 440-460: 3%, 460-480: 1%, 480-500: 5%, 500-520: 10%, 520-540: 1%, 540-560: 4%, 560-580: 8%, 580-600: 9%, 600-900: 0%

time. The loss process was moved from a setting of 0% loss to 5% loss at time 200 seconds, from 5% loss to 10% loss at 300 seconds, and finally from 10% loss back to 0% loss at 400 seconds. We did not obtain results for the Android WiFi device for Scenarios 3 or 4. Looking at the time interval 200 to 300 seconds in Figures 5a through 5d, we observe that Netflix moves to a steady state throughput ranging from 1.40 Mbps to 2.20 Mbps depending on the device. Repeating our *iperf* test, we observe that a backlogged TCP flow achieved an average throughput of 1.91 Mbps (standard deviation of 0.08 Mbps) when the flow was subject to 5% packet loss. When the packet loss rate was 10% we observe the Netflix clients move to a steady state throughput that varies from 0.49 Mbps to 1.10 Mbps depending on the device. Our *iperf* test shows a TCP throughput of 0.60 Mbps (with a standard deviation of 0.05 Mbps) when the flow is subject to 10% packet loss. These results suggest that the TCP congestion control algorithm is the dominant control algorithm during periods of heavy, sustained network congestion.

D. Scenario 4 results

Figure 6 illustrates the results associated with Measurement Scenario 4 where the Netflix session traffic was subject to a non-stationary loss process that evolved quickly over time. In the time interval 300 to 600 seconds, the loss process moved to a new state (i.e., we changed the mean loss rate parameter used by the uncorrelated packet loss process created by *netem*) every 20 seconds. The specific transition times and loss rate settings are identified in the caption of Figure 6. The average loss rate of the 17 settings was 4.4%. This scenario was used to explore the stability of the Netflix adaptation algorithm when subject to 'chaotic' (i.e., non-stationary) packet loss behavior. As expected, we did not see Netflix reach a steady state in this time interval. The average bandwidth consumed by the Netflix session during the time interval ranged from 1.81 to 2.75 Mbps depending on the device. We ran the *iperf* test using the same sequence of loss rate transitions. *iperf* suggests that TCP would obtain an average throughput of 20.5 Mbps during the 300 second period of variable packet loss. This is quite different from what we observe with Netflix. The random loss pattern includes two back-to-back intervals that set the loss rate to 0% allowing the backlogged TCP connection throughput to reach 100 Mbps for about 40 seconds during the 300 second time period. These results suggest that in certain situations (such as when network conditions are highly variable) the application level control and TCP control co-exist in a manner that potentially reduces the flow's fair share of available bandwidth with the hope that shielding the end user from short term network congestion fluctuation improves the end user's perceived quality.

V. Related Work

As summarized in [5], the DASH standards focus primarily on the format of data by defining the syntax required to describe content manifests and segments containing actual chunks of encoded video streams. The standards do not identify specific encoding methods, specific protocol behaviors such as how frequently the client should request data, or system guidelines such as optimal choices of segment size.

Academic research on DASH is just emerging. We summarize the related work by identifying three distinct categories of published research: measurement-based performance studies, algorithm enhancements, and performance or quality metric

development.

The work in [6], [7], [8], [9], [10] have established the basic behaviors of DASH applications. The work in [6] is the most similar to our measurement study. They used *dummynet* to provide a controlled network testbed. While we primarily controlled the loss rate observed by the Netflix sessions, the authors in [6] controlled the emulated access link bandwidth (referred to as *avail-bw*) to the clients under observation. They evaluated the three players when the *avail-bw* was unconstrained, constrained to 5 Mbps, or subject to periodic positive or negative spikes in *avail-bw*. Their results show differences in the details of each client's response to change (positive or negative) in available bandwidth. The work in [11], [10] discusses the benefits of parallel TCP connections although the authors point out the potential unfairness. The work in [8] shows performance drops if HTTP persistent connections are not used. The work in [9] studied Netflix and YouTube streaming using a variety of client devices. They show that YouTube and Netflix each can exhibit widely differing ON/OFF cycle behaviors ranging from always sending (no ON/OFF cycles), to frequent bursts of small amounts of data (e.g., 64 KB), to infrequent bursts of large amounts of data (> 2.5 MB). The exact behavior depends on the network properties (bottleneck link rates) and the capabilities of the client device. The work in [7] provides more insight in the ON/OFF behavior of YouTube and suggests the relatively bursty nature of the YouTube server might contribute to higher than necessary packet loss rates.

The work in [12], [13], [14], [15], [16] consider possible improvements to DASH-based streaming. [12] considers the benefits of using Scalable Video Coding (SVC) outweigh the additional overhead. The authors identify optimal approaches for determining which block should be selected next. The block might be data that will improve the video quality of a frame whose base layer has already arrived or it might be a block corresponding to the base layer of a frame further out in time). The work in [16] shows that DASH applications extended to support SVC can improve cache hit rates in typical CDN distribution networks. The work in [13], [15], [14] focus on the adaptation algorithm. All approaches seek to react to persistent impairment by dropping to lower bitrate content and then restore to higher quality content as conditions improve. The challenge is to do this so as perceived quality is maximized.

VI. Conclusions

Video streaming has been widely deployed and studied for decades. However DASH-based streaming is very different from UDP-based streaming as it involves adaptation both by the application and by TCP. The dynamics and implications of multiple levels of end-to-end congestion control are not well understood. Our research provides greater insight into this unexplored problem. We summarize the results as follows:

- The basic response to network congestion is similar across the devices that were studied. However, the steady state bandwidth consumption achieved by different devices during periods of sustained congestion varied as did the details of how the client consumed available bandwidth following periods of congestion.
- The playback buffer sizes ranged from 30 seconds (on the Android WiFi device) to 4 minutes (on the Windows Wired device). The size of the playback buffer is crucial in masking the effects of network congestion from the perceived quality.The average size of the client request ranged from 1.8 MB to 3.57 MB (we did not show these results in the paper).
- The Netflix adaptation appears to default to TCP control during periods of heavy, sustained network congestion. However, the application algorithm is clearly intertwined with TCP control during periods of volatile network conditions.

From a network operator's perspective, Netflix is a very well behaved application. Unfortunately, assessing the application performance of video streaming is challenging since the details of the video encoding and decoding process significantly impact the perceived quality, especially when the streams are subject to network impairment. We plan to address this issue in future work by incorporating in our methods human perceived quality assessments.

References

[1] ISO/IEC, "Information technology – mpeg systems technologies – part 6: Dynamic adaptive streaming over http (dash)," http://mpeg.chiariglione.org/working_documents/mpeg-dash/dash-dis.zip, January 2011.

[2] T. Stockhammer, "Dynamic adaptive streaming over http –: standards and design principles," in *Proceedings of the second annual ACM conference on Multimedia systems*, ser. MMSys '11. New York, NY, USA: ACM, 2011, pp. 133–144. [Online]. Available: http://doi.acm.org/10.1145/1943552.1943572

[3] 3GPP TS 26.247 version 10.1.0 Release 10, "Transparent end-to-end packet switched streaming service (pss); progressive download and dynamic adaptive service over http," http://www.etsi.org/deliver/etsi_ts/126200_126299/126247/10.01.00_60/ts_126247v100100p.pdf, January 2012.

[4] V. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z.-L. Zhang, "Unreeling netflix: Understanding and improving multi-cdn movie delivery," in *INFOCOM, 2012 Proceedings IEEE*, A. G. Greenberg and K. Sohraby, Eds. IEEE, Mar 2012, pp. 1620–1628. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6195531

[5] I. Sodagar, "The mpeg-dash standard for multimedia streaming over the internet," *IEEE MultiMedia*, vol. 18, no. 4, pp. 62–67, Oct. 2011. [Online]. Available: http://dx.doi.org/10.1109/MMUL.2011.71

[6] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http," in *Proceedings of the second annual ACM conference on Multimedia systems*, ser. MMSys '11. New York, NY, USA: ACM, 2011, pp. 157–168. [Online]. Available: http://doi.acm.org/10.1145/1943552.1943574

[7] S. Alcock and R. Nelson, "Application flow control in youtube video streams," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 2, pp. 24–30, Apr. 2011. [Online]. Available: http://doi.acm.org/10.1145/1971162.1971166

[8] S. Lederer, C. Müller, and C. Timmerer, "Dynamic adaptive streaming over http dataset," in *Proceedings of the 3rd Multimedia Systems Conference*, ser. MMSys '12. New York, NY, USA: ACM, 2012, pp. 89–94. [Online]. Available: http://doi.acm.org/10.1145/2155555.2155570

[9] A. Rao, A. Legout, Y.-s. Lim, D. Towsley, C. Barakat, and W. Dabbous, "Network characteristics of video streaming traffic," in *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, ser. CoNEXT '11. New York, NY, USA: ACM, 2011, pp. 25:1–25:12. [Online]. Available: http://doi.acm.org/10.1145/2079296.2079321

[10] R. Kuschnig, I. Kofler, and H. Hellwagner, "Evaluation of http-based request-response streams for internet video streaming," in *Proceedings of the second annual ACM conference on Multimedia systems*, ser. MMSys '11. New York, NY, USA: ACM, 2011, pp. 245–256. [Online]. Available: http://doi.acm.org/10.1145/1943552.1943585

[11] ——, "Improving internet video streamilng performance by parallel tcp-based request-response streams," in *Proceedings of the 7th IEEE conference on Consumer communications and networking conference*, ser. CCNC'10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 200–204. [Online]. Available: http://dl.acm.org/citation.cfm?id=1834217.1834257

[12] T. Andelin, V. Chetty, D. Harbaugh, S. Warnick, and D. Zappala, "Quality selection for dynamic adaptive streaming over http with scalable video coding," in *Proceedings of the 3rd Multimedia Systems Conference*, ser. MMSys '12. New York, NY, USA: ACM, 2012, pp. 149–154. [Online]. Available: http://doi.acm.org/10.1145/2155555.2155580

[13] A. C. Begen and K. Mayer-Patel, Eds., *Proceedings of the Second Annual ACM SIGMM Conference on Multimedia Systems, MMSys 2011, Santa Clara, CA, USA, February 23-25, 2011*. ACM, 2011.

[14] K. Evensen, D. Kaspar, C. Griwodz, P. Halvorsen, A. F. Hansen, and P. Engelstad, "Improving the performance of quality-adaptive video streaming over multiple heterogeneous access networks," in *RFID Explained'11*, 2011, pp. 57–68.

[15] C. Liu, I. Bouazizi, and M. Gabbouj, "Rate adaptation for adaptive http streaming," in *Proceedings of the second annual ACM conference on Multimedia systems*, ser. MMSys '11. New York, NY, USA: ACM, 2011, pp. 169–174. [Online]. Available: http://doi.acm.org/10.1145/1943552.1943575

[16] Y. Sánchez de la Fuente, T. Schierl, C. Hellge, T. Wiegand, D. Hong, D. De Vleeschauwer, W. Van Leekwijck, and Y. Le Louédec, "idash: Improved dynamic adaptive streaming over http using scalable video coding," in *Proceedings of the second annual ACM conference on Multimedia systems*, ser. MMSys '11. New York, NY, USA: ACM, February 2011, pp. 257–264. [Online]. Available: http://doi.acm.org/10.1145/1943552.1943586