Building Reproducible Video Streaming Traffic Generators

Calvin Ardi
USC/ISI
Marina del Rey, CA, USA
calvin@isi.edu

Alefiya Hussain
USC/ISI
Marina del Rey, CA, USA
hussain@isi.edu

Stephen Schwab
USC/ISI
Marina del Rey, CA, USA
schwab@isi.edu

ABSTRACT

Video streaming traffic dominates Internet traffic. However, there is a dearth of tools to generate such traffic on emulation-based testbeds. In this paper we present tools to create representative and reproducible video streaming traffic to evaluate the next generation of traffic classification, Quality of Service (QoS) algorithms and traffic engineering systems. We discuss 27 different combinations of streaming video traffic types in this preliminary work, and illustrate the diversity of network-level dynamics in these protocols.

ACM Reference Format:
Calvin Ardi, Alefiya Hussain, and Stephen Schwab. 2021. Building Reproducible Video Streaming Traffic Generators. In *Cyber Security Experimentation and Test Workshop (CSET '21), August 9, 2021, Virtual, CA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3474718.3474721

1 INTRODUCTION

Video streaming traffic has and continues to be the majority of Internet traffic, accounting for roughly 58 % of global Internet use (2020, [14]) and is predicted to be 82 % of all traffic by 2022 [6]. Delivering video across the Internet is supported by a diverse set of infrastructure, client/server software, encoding algorithms, and network protocols. Although many parts of the end-to-end video delivery on the Internet has been studied in great detail [1, 3, 9, 11], there are no readily available tools for video traffic to support testbed-based experiments. Prior work in traffic generation and simulation typically focus on maximizing throughput for benchmarks and stress testing [12] or on simulating the behavior of the underlying protocols (TCP, UDP, and others) [2, 15, 16].

While experimenters can iterate on and test their network system components over the Internet, often coupling them with popular content providers, there are a lack of tools to support principled experimentation with video traffic on testbeds and the Internet. Experimenters need these video traffic tools in order to evaluate the next generation of systems for network traffic classification and engineering and QoS algorithms. These tools should allow experimenters to test and evaluate these systems and their individual components during its development, and include both client and server endpoints used for watching and delivering video. These tools should also send traffic responsibly on the Internet, in order to minimize impact on third-party services.

CSET '21, August 9, 2021, Virtual, CA, USA
© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9065-1/21/08…$15.00
https://doi.org/10.1145/3474718.3474721

In this paper, we present our preliminary work towards building video streaming traffic generators that are both representative and reproducible. We implement in our generators both the client and server endpoints, and each individual endpoint can be used on its own to test additional components within the network. Our generators produce *representative* traffic on-the-wire by using freely available or permissively-licensed videos and streaming them across a variety of transport protocols. We enable *reproducible* experiments by emulating the process of "watching" videos with a systematic and well-defined methodology. We discuss in § 2 current state-of-art video streaming protocols and how they can be used to recreate representative and reproducible scenarios on an emulation-based testbed. Our tools support 27 video streaming protocol and software combinations, and can be fully automated. Further, they can be configured to run completely end-to-end within a self-contained environment on the testbed or reach-out to remote servers and services over the Internet to provide even more realism.

We discuss in § 3 the varied network-level traffic patterns generated from a sample of the 27 protocols on an emulation-based testbed. We analyze the underlying video streaming performance and behavior on both a local network and the Internet, highlighting the differences in bandwidth throughput profiles between different streaming and transport protocols. Our goal is to enable representative and reproducible experiments for video streaming traffic and this initial set of generators provides a wide spectrum of protocols. We make our generators and tools available at https://mergetb.org/projects/searchlight/ in order to enable research in this domain.

2 REPRODUCIBLE VIDEO STREAMING TRAFFIC GENERATORS

In this section, we discuss the design decisions for representative protocols and software to enable reproducible experiments on emulation-based testbeds.

2.1 Using Representative Protocols

There are many formats and protocols for video streaming. We consider 27 combinations, that include different streaming protocols, the encapsulation format for video streaming, and "transport" protocols, the underlying application-level protocols that move data between client and server. We focus only on desktop browsers in this paper, and plan on addressing mobile and other application-specific clients as future work.

[0]This work was sponsored by Sandia National Laboratory (SNL) under PO2160586. This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of Sandia National Laboratory, the Department of Defense, or the U.S. Government.

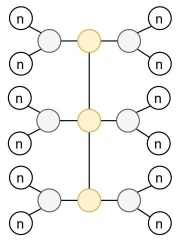


Figure 1: Network Topology

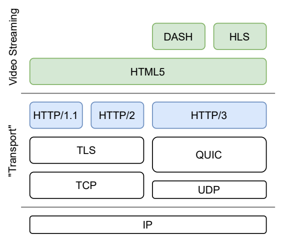


Figure 2: Protocols

Streaming protocols. We focus on Dynamic Adaptive Streaming over HTTP (DASH), HTTP Live Streaming (HLS), and basic HTML5 video as our representative streaming protocols. While there are many different streaming protocols available, prior work [1] has found that DASH and HLS accounted for 83 % of video playback time (2018), thus making the inclusion of both protocols in our initial work a straightforward choice.

We additionally add HTML5 video to act as a baseline for evaluation and comparison. Although HTML5 video lacks many of the complexities and advances in streaming technology (discussed next), it is broadly supported in popular web browsers and easily included by website authors. We plan to include other streaming protocols not mentioned here as future work.

For better reproducibility, we fix the video resolution for the entire playback duration to 480p, 720p, or 1080p. Although the primary benefit of using Adaptive Bitrate (ABR) streaming protocols (DASH, HLS) is their ability to dynamically switch resolutions based on network conditions, our initial goal is to generate consistent, and thus reproducible, traffic on the network. Fixed playback resolution, or Constant Bitrate (CBR) streaming, is the default behavior of HTML5 video. We plan to provide a wider range of video resolutions as well as resolution switching that is both dynamic and reproducible.

Transport protocols. Our selection of streaming protocols (DASH, HLS, HTML5) operates over HTTP (shown in Fig. 2). We have considered other protocols like RTMP, but leave its support for future work: prior work found that RTMP and others [1] are not widely deployed today.

We support for multiple versions of HTTP, including HTTP/1.1, HTTP/2, and HTTP/3 (currently an IETF draft standard) both in cleartext when possible and with encryption. Table 1 shows the various software and supported HTTP combinations and Fig. 2 shows both the relationship and choices between the streaming (green) and transport (blue) protocols. The IETF QUIC Working Group is actively finalizing standards for HTTP/3 [4] and QUIC [10], both of which offer benefits in more effective bandwidth use, security, and flexibility in protocol evolution. As HTTP/3's prevalence continues to grow, our tools will enable us to test our algorithms and systems with realistic traffic and in a controlled, systematic manner.

We also support cleartext options when possible. While the Internet is moving towards opportunistically encrypting data-in-transit (for example, HTTP/3 will always encrypt), using cleartext has been useful for debugging and protocol analysis. In some cases, either the server or client software does not implement a particular feature of HTTP. For example, while HTTP/2 cleartext is implemented in some web servers, the major browsers do not support it.

Table 1: Supported Protocol and Software Combinations

software	type	HTTP/1.1		HTTP/2		HTTP/3
		clear	+ TLS	clear	+ TLS	TLS
Chrome	client	●	●	○	●	●
Firefox	client	●	●	○	●	●
Caddy	server	●	●	●	●	◑
nginx[a]	server	●	●	●	●	○
Apache2[a]	server	●	●	●	●	○

Implementation: ● = full, ◑ = partial, ○ = none [a] for comparison purposes

2.2 Enabling Reproducible Experiments

We are building our video traffic generators to enable reproducible experiments for building and testing traffic engineering systems, and client or server video applications. Our generators can also be used to generate traffic in a self-contained manner, within a testbed, or to remote, Internet services.

Our traffic generators provide the end-to-end application traffic necessary for traffic engineering systems. Using simple and well-defined generators enable us to create reproducible experiments that system builders can use to validate and rapidly iterate over their designs.

We build simplified, static websites for our video streaming traffic generators to maximize the signal-to-noise ratio on the wire. We focus on the video streaming content and exclude most of the framing support that generally surrounds the main content on today's webpages. While content like ads, comments, or other dynamically loaded sections ("Coming Up Next", etc.) are prevalent today, they add an additional layer of complexity that is not needed at early stages in the design process. Our design has enabled others to focus their efforts on prototyping and deploying an early traffic engineering system without other irregularities—later, we plan to introduce additional options in our servers to increase the complexity and realism that is more reflective of today's websites.

Finally, our traffic generators are designed to be self-contained within a testbed environment or used with live Internet services. We currently limit using our generators on Internet services, taking care to remain within the typical Internet noise threshold. We plan to carefully explore how we can emulate large amounts of traffic on the Internet without undue impact to other users and services.

2.3 Implementation

We implement our video streaming traffic generators using the Playwright [13] browser automation framework to coordinate and script the client behavior and the Caddy [5] web server with configurations that control and implement our chosen protocols (§ 2.1).

For rapid prototyping and testing, we instrument and automate the web browser as our video application client. Using the Playwright framework, we instrument the Chrome and Firefox web browsers using JavaScript to emulate an end-user's simple behavior of watching video. For example, we have several automations that emulate a user watching a video for a specified amount of time on both local and remote services (like YouTube): our scripts will first navigate to a specified internal or external URL, select the specified resolution, press "play" and "watch" the video, and finally close the browser. To retrieve ground truth, we also capture network-

(PCAPs) and HTTP application-level traffic for both online and offline analysis.

On the server-side, we use the Caddy web server as it supports a variety of HTTP protocols (§ 2.1) and is easily extensible for our monitoring infrastructure (not covered in this paper). For easier analysis and validation on the wire, we build simple, static webpages incorporating dash.js (DASH, [8]), HLS.js (HLS, [7]), or plain HTML5 for videos, and serve them at unique URL endpoints. For example, 1080p video playback using DASH is available at `https://example.internal/dash-1080p.htm`, and so on. To further increase performance, we also pre-compute video content into relevant formats and chunks for ABR streaming as opposed to dynamically transcoding on-demand.

We use and support our implementation on emulation-based Linux testbeds and standalone computers (Linux, macOS), with support on Microsoft Windows planned. Our traffic generators, tools, and analysis are available at https://mergetb.org/projects/searchlight/.

3 EXPERIMENTATION

In this section, we present a series of initial experiments designed to demonstrate our traffic generators and illustrate some of the underlying video streaming behavior on the network. We use our experiments to understand the dynamics on the network for the various streaming protocols (§ 3.1), how our video streaming traffic generators handle multiple clients (§ 3.2), and the network throughput between different HTTP versions (§ 3.3).

3.1 Differences between Video Streaming Protocols

Our experiments indicate that each video streaming protocol has unique bandwidth-throughput characteristics at the network level. We run our traffic generators on one Google Chrome client, connecting directly to a server and watching video over 3 streaming protocols at 1080p for roughly 90–120 s on a 10 Mbps link.

Fig. 3 shows three graphs of data transferred over time, binned by 1 s. Each graph contains a line for each TCP connection (most browsers support up to six parallel connections in HTTP/1.1) and the top-most line (sometimes overlapping with other lines) is the sum across all connections. Once we start playing video at roughly 5–15 s, we see that each streaming protocol can effectively use all available bandwidth. We observe that DASH (Fig. 3b) and HLS (Fig. 3c) use bandwidth more efficiently as shown in the various "valleys" in each line: this behavior is likely due to a more intelligent buffering strategy.

DASH also takes advantage of multiple connections by streaming audio and video in separate connections, as seen in Fig. 3b. The DASH protocol separates audio and video file chunks, in contrast to HTML5 and HLS, in which each file or file chunk contains both, respectively. Initially, connection '1' downloads video and '2' downloads audio chunks, switching once at roughly 30 s. We do not have an explanation for why a connection switches at this point, but hypothesize that connection management decisions are made at the underlying browser-level rather than at the video player.

Finally, HLS in Fig. 3c seems to make the most efficient use of bandwidth and number of connections. Both the supporting content (HTML, CSS, JavaScript) and video content are handled with only one connection ('0'). In contrast to DASH, video content chunks for HLS contain both audio and video segments (and thus would not necessarily benefit from parallel connections). We believe that connection '1' was opened (and subsequently closed shortly after) as a browser-level performance optimization.

Although not shown here, we have also observed that DASH and HLS are capable of buffering enough video to continue playback against a temporary connection disruption. We did not observe this behavior with HTML5. We next look at the profiles of video streaming on different HTTP versions.

3.2 Handling Multiple Client Generators

We look at the network throughput profiles of running multiple generators simultaneously. Our setup is the same as in § 3.1, except with eight Google Chrome clients streaming from one server for 250 s on a 100 Mbps link: the network topology is shown in Fig. 1, with clients and server located on the nodes labeled 'n'.

Fig. 4 shows three graphs of data transferred over time, binned by 1 s. Each graph contains a line for each of the eight client-server node pairs, and the top-most line is the sum across all pairs.

We see that the total bandwidth use over time across all three protocols initially maxes out the link at 100 Mbps, then exhibits steady-state behavior similar to what we saw earlier in § 3.1, with periodic spikes. We focus on the total throughput profile as the individual profile of any one client-server pair is hard to distinguish.

The differences at the start of each graph in Fig. 4 show that the initial buffer fill behavior is much more varied across protocols when multiple client generators are used. HTML5 (Fig. 4a) uses all the available bandwidth for 30 s before decreasing its use. DASH (Fig. 4b) and HLS (Fig. 4c) sustain maximum bandwidth for 90 s and 110 s, respectively.

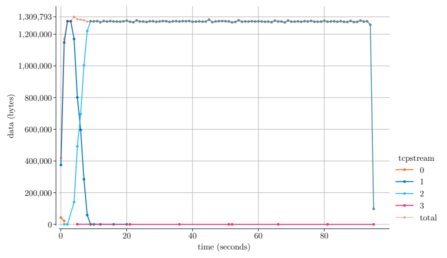
The difference in initial throughput between HTML5 and ABR protocols (DASH, HLS) is again likely due to the difference in buffer filling strategies, with HTML5 keeping a minimal playback buffer (we observed earlier that HTML5 suffers early on from a disrupted connection). While the available bandwidth to any one client is limited due to contention, it is sufficient to maintain continuous playback. Once all clients have reached steady-state in ABR streaming, we see the usual bursty behavior as each client can quickly refill its buffer due to the lack of contention.

Finally, we note that we are not necessarily able to observe the video streaming Quality of Experience (QoE) with throughput profiles alone. For example, one of our eight clients might experience more rebuffering events, with slight pauses in playback, than others as that client constantly attempts to retrieve more data. We plan on developing instrumentation for client browsers to report QoE measurement data as future work.

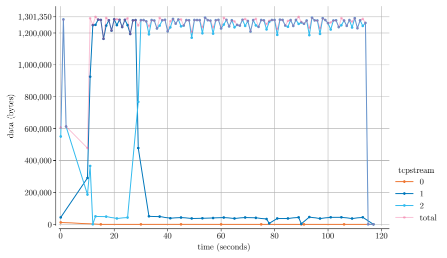
3.3 Differences between HTTP Versions

Lastly, we look at the network throughput profiles of various transport protocols on a remote video streaming service. We run our traffic generators on one Google Chrome client, connecting to YouTube and watching video over 3 different HTTP versions, all encrypted under TLS, at 1440p for roughly 80 s on a 200 Mbps link.

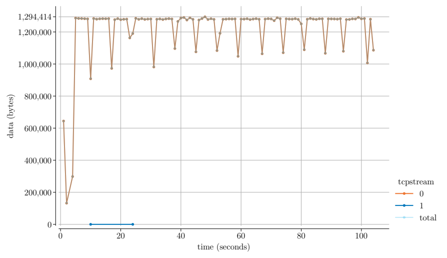
Fig. 5 shows three graphs of data transferred over time, binned by 1 s. While our client connects to multiple servers for supporting



(a) HTML5

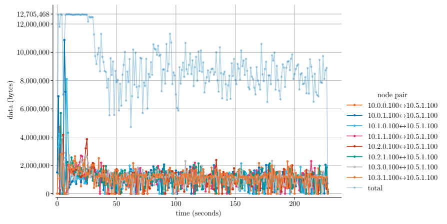


(b) DASH

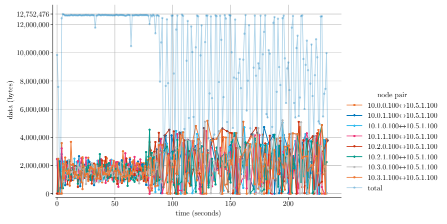


(c) HLS

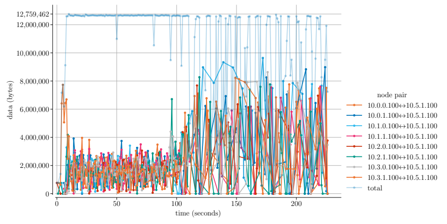
Figure 3: Streaming video from one server to one client over multiple video streaming protocols via HTTP/1.1 cleartext, 1080p resolution, 10 Mbps link speed.



(a) HTML5

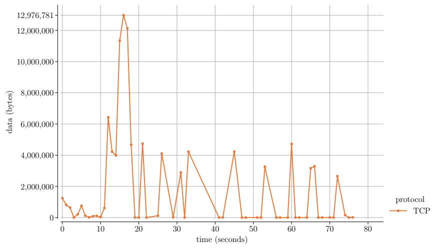


(b) DASH

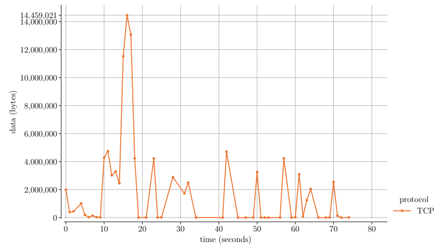


(c) HLS

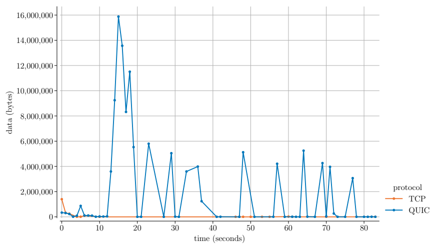
Figure 4: Streaming video from one server to eight clients over multiple video streaming protocols via HTTP/1.1 cleartext, 1080p resolution, 100 Mbps link speed.



(a) HTTP/1.1 + TLS



(b) HTTP/2 + TLS



(c) HTTP/3 (Q050)

Figure 5: Streaming video from YouTube to one client over multiple transport protocols, 1440p resolution, 200 Mbps link speed.

content (ads, telemetry, and CDNs), we graph the total throughput per underlying protocol (TCP or QUIC) for simplicity. We start video playback at around 10 s into the capture to allow enough time to download the website's supporting content. We observe in all three protocols similar playback behavior in our prior experiments: an initial spike in throughput usage to fill the buffer and compute available bandwidth, with steady-state bursty behavior following.

While HTTP/1.1 (Fig. 5a) and HTTP/2 (Fig. 5b) have roughly the same throughput profile, we see that YouTube leverages HTTP/3, if available, with an overall throughput that is slightly higher. We see in Fig. 5c an initial TCP connection, which negotiates and switches over to the use of HTTP/3 over QUIC for the rest of the experiment duration—the lingering TCP connection remains open, but unused, with keepalives. We also observe a larger maximum throughput usage of 128 Mbps compared to earlier HTTP versions (104–116 Mbps). We plan to continue adding support for more remote Internet services and HTTP/3.

4 FUTURE WORK AND CONCLUSION

In this paper we presented tools to create representative and reproducible video streaming traffic to evaluate next generation QoS algorithms and traffic classification and engineering systems. We discussed 27 different combinations of streaming video traffic types in this preliminary work, and illustrated the diversity of network-level dynamics in these protocols. We will continue to build our traffic generators with the goal of scaling up on the order of 10^3 clients and servers. To make traffic more representative, we plan to build models of human behavior in video streaming consumption (binge watching vs. skipping around) and use these models to drive our clients. On the network- and server-side, we plan to emulate realistic network architectures, like CDNs and anycast, and implement more complex websites, incorporating supporting frame content and ABR streaming behavior. Our traffic generators are available at https://mergetb.org/projects/searchlight/.

REFERENCES

[1] Zahaib Akhtar, Yun Seong Nam, Jessica Chen, Ramesh Govindan, Ethan Katz-Bassett, Sanjay Rao, Jibin Zhan, and Hui Zhang. 2018. Understanding Video Management Planes. In *Proceedings of the Internet Measurement Conference 2018* (Boston, MA, USA) *(IMC '18)*. Association for Computing Machinery, New York, NY, USA, 238–251. https://doi.org/10.1145/3278532.3278554

[2] Doreid Ammar, Thomas Begin, and Isabelle Guerin-Lassous. 2011. A New Tool for Generating Realistic Internet Traffic in NS-3. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques* (Barcelona, Spain) *(SIMUTools '11)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, BEL, 81–83.

[3] Abdelhak Bentaleb, Bayan Taani, Ali C. Begen, Christian Timmerer, and Roger Zimmermann. 2019. A Survey on Bitrate Adaptation Schemes for Streaming Media Over HTTP. *IEEE Communications Surveys Tutorials* 21, 1 (2019), 562–585. https://doi.org/10.1109/COMST.2018.2862938

[4] Mike Bishop. 2021. *Hypertext Transfer Protocol Version 3 (HTTP/3)*. Internet-Draft draft-ietf-quic-http-34. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34 Work in Progress.

[5] Caddy. 2020. Caddy v2.1.1. https://caddyserver.com/

[6] Cisco. 2018. Cisco Visual Networking Index: Forecast and Trends, 2017–2022 White Paper. (27 Nov. 2018). https://web.archive.org/web/20200215211855/https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html

[7] Dailymotion. 2021. HLS.js v1.0.3. https://github.com/video-dev/hls.js

[8] DASH Industry Forum. 2021. dash.js JavaScript Reference Client v3.2.2. https://reference.dashif.org/dash.js/

[9] Mojgan Ghasemi, Partha Kanuparthy, Ahmed Mansy, Theophilus Benson, and Jennifer Rexford. 2016. Performance Characterization of a Commercial Video Streaming Service. In *Proceedings of the 2016 Internet Measurement Conference* (Santa Monica, California, USA) *(IMC '16)*. Association for Computing Machinery, New York, NY, USA, 499–511. https://doi.org/10.1145/2987443.2987481

[10] Jana Iyengar and Martin Thomson. 2021. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000. https://doi.org/10.17487/RFC9000

[11] Sina Keshvadi and Carey Williamson. 2021. An Empirical Measurement Study of Free Live Streaming Services. In *Passive and Active Measurement*, Oliver Hohlfeld, Andra Lutu, and Dave Levin (Eds.). Springer International Publishing, Cham, 111–127. https://doi.org/10.1007/978-3-030-72582-2_7

[12] ESnet / Lawrence Berkeley National Laboratory. 2020. iperf3 v3.9. https://software.es.net/iperf/

[13] Microsoft. 2021. Playwright v1.10.0. https://playwright.dev/

[14] Sandvine. 2020. The Global Internet Phenomena Report COVID-19 Spotlight. (7 May 2020). https://www.sandvine.com/phenomena

[15] Joel Sommers, Hyungsuk Kim, and Paul Barford. 2004. Harpoon: A Flow-Level Traffic Generator for Router and Network Tests. *SIGMETRICS Perform. Eval. Rev.* 32, 1 (June 2004), 392. https://doi.org/10.1145/1012888.1005733

[16] Michele C. Weigle, Prashanth Adurthi, Félix Hernández-Campos, Kevin Jeffay, and F. Donelson Smith. 2006. Tmix: A Tool for Generating Realistic TCP Application Workloads in Ns-2. *SIGCOMM Comput. Commun. Rev.* 36, 3 (July 2006), 65–76. https://doi.org/10.1145/1140086.1140094